PyTorch

Release Notes

RN-08516-001_v26.11 | November 2026

Table of Contents

Chapter 1. PyTorch Overview 1
Chapter 2. Pulling A Container 2
Chapter 3. Running PyTorch 3
Chapter 4. PyTorch Release 26.11 5
Chapter 5. PyTorch Release 26.10 15
Chapter 6. PyTorch Release 26.09 25
Chapter 7. PyTorch Release 26.08 35
Chapter 8. PyTorch Release 26.07 45
Chapter 9. PyTorch Release 26.06 54
Chapter 10. PyTorch Release 26.05 63
Chapter 11. PyTorch Release 26.04 72
Chapter 12. PyTorch Release 26.03 81
Chapter 13. PyTorch Release 26.02 90
Chapter 14. PyTorch Release 26.01 99
Chapter 15. PyTorch Release 25.12 108
Chapter 16. PyTorch Release 25.11 117
Chapter 17. PyTorch Release 25.10 126
Chapter 18. PyTorch Release 25.09 134
Chapter 19. PyTorch Release 25.08 142
Chapter 20. PyTorch Release 25.06 150
Chapter 21. PyTorch Release 25.05 158
Chapter 22. PyTorch Release 25.04 166
Chapter 23. PyTorch Release 25.03 174
Chapter 24. PyTorch Release 25.02 182
Chapter 25. PyTorch Release 25.01 190
Chapter 26. PyTorch Release 24.12 198
Chapter 27. PyTorch Release 24.11 206
Chapter 28. PyTorch Release 24.10 214
Chapter 29. PyTorch Release 24.09 222

Chapter 30. PyTorch Release 24.08....232
Chapter 31. PyTorch Release 24.07....242
Chapter 32. PyTorch Release 24.06....252
Chapter 33. PyTorch Release 24.05....260
Chapter 34. PyTorch Release 24.04....268
Chapter 35. PyTorch Release 24.03....276
Chapter 36. PyTorch Release 24.02....284
Chapter 37. PyTorch Release 24.01....292
Chapter 38. PyTorch Release 23.12....300
Chapter 39. PyTorch Release 23.11....308
Chapter 40. PyTorch Release 23.10....316
Chapter 41. PyTorch Release 23.09....324
Chapter 42. PyTorch Release 23.08....332
Chapter 43. PyTorch Release 23.07....340
Chapter 44. PyTorch Release 23.06....348
Chapter 45. PyTorch Release 23.05....356
Chapter 46. PyTorch Release 23.04....364
Chapter 47. PyTorch Release 23.03....372
Chapter 48. PyTorch Release 23.02....380
Chapter 49. PyTorch Release 23.01....388
Chapter 50. PyTorch Release 22.12....396
Chapter 51. PyTorch Release 22.11....404
Chapter 52. PyTorch Release 22.10....412
Chapter 53. PyTorch Release 22.09....419
Chapter 54. PyTorch Release 22.08....427
Chapter 55. PyTorch Release 22.07....434
Chapter 56. PyTorch Release 22.06....441
Chapter 57. PyTorch Release 22.05....448
Chapter 58. PyTorch Release 22.04....455
Chapter 59. PyTorch Release 22.03....462
Chapter 60. PyTorch Release 22.02....469

Chapter 61. PyTorch Release 22.01 476
Chapter 62. PyTorch Release 21.12 483
Chapter 63. PyTorch Release 21.11 490
Chapter 64. PyTorch Release 21.10 497
Chapter 65. PyTorch Release 21.09 503
Chapter 66. PyTorch Release 21.08 509
Chapter 67. PyTorch Release 21.07 515
Chapter 68. PyTorch Release 21.06 521
Chapter 69. PyTorch Release 21.05 527
Chapter 70. PyTorch Release 21.04 533
Chapter 71. PyTorch Release 21.03 539
Chapter 72. PyTorch Release 21.02 545
Chapter 73. PyTorch Release 21.01 551
Chapter 74. PyTorch Release 20.12 552
Chapter 75. PyTorch Release 20.11 558
Chapter 76. PyTorch Release 20.10 564
Chapter 77. PyTorch Release 20.09 570
Chapter 78. PyTorch Release 20.08 576
Chapter 79. PyTorch Release 20.07 582
Chapter 80. PyTorch Release 20.06 588
Chapter 81. PyTorch Release 20.03 595
Chapter 82. PyTorch Release 20.02 601
Chapter 83. PyTorch Release 20.01 607
Chapter 84. PyTorch Release 19.12 613
Chapter 85. PyTorch Release 19.11 618
Chapter 86. PyTorch Release 19.10 623
Chapter 87. PyTorch Release 19.09 628
Chapter 88. PyTorch Release 19.08 633
Chapter 89. PyTorch Release 19.07 638
Chapter 90. PyTorch Release 19.06 642
Chapter 91. PyTorch Release 19.05 646

Chapter 92. PyTorch Release 19.04 650
Chapter 93. PyTorch Release 19.03 654
Chapter 94. PyTorch Release 19.02 657
Chapter 95. PyTorch Release 19.01 660
Chapter 96. PyTorch Release 18.12 663
Chapter 97. PyTorch Release 18.11 665
Chapter 98. PyTorch Release 18.10 667
Chapter 99. PyTorch Release 18.09 669
Chapter 100. PyTorch Release 18.08 672
Chapter 101. PyTorch Release 18.07 674
Chapter 102. PyTorch Release 18.06 676
Chapter 103. PyTorch Release 18.05 678
Chapter 104. PyTorch Release 18.04 680
Chapter 105. PyTorch Release 18.03 681
Chapter 106. PyTorch Release 18.02 682
Chapter 107. PyTorch Release 18.01 684
Chapter 108. PyTorch Release 17.12 685
Chapter 109. PyTorch Release 17.11 686
Chapter 110. PyTorch Release 17.10 688
Chapter 111. PyTorch Release 17.09 689
Chapter 112. PyTorch Release 17.07 691
Chapter 113. PyTorch Release 17.06 692
Chapter 114. PyTorch Release 17.05 693
Chapter 115. PyTorch Release 17.04 694

Chapter 1. PyTorch Overview

The NVIDIA® Deep Learning SDK accelerates widely-used deep learning frameworks such as PyTorch.

PyTorch is a GPU-accelerated tensor computational framework with a Python front end. Functionality can be easily extended with common Python libraries such as NumPy, SciPy, and Cython. Automatic differentiation is done with a tape-based system at both a functional and neural network layer level. This functionality brings a high level of flexibility and speed as a deep learning framework and provides accelerated NumPy-like functionality.

PyTorch also includes standard defined neural network layers, deep learning optimizers, data loading utilities, and multi-gpu, and multi-node support. Functions are executed immediately instead of enqueued in a static graph, improving ease of use and provides a sophisticated debugging experience.

In the container, see `/workspace/README.md` for information about customizing your PyTorch image. For more information about PyTorch, including tutorials, documentation, and examples, see:

- PyTorch website
- PyTorch project

This document provides information about the key features, software enhancements and improvements, known issues, and how to run this container.

Chapter 2. Pulling A Container

About this task

Before you can pull a container from the NGC container registry:

- Install Docker.
  - For NVIDIA DGX™ users, see Preparing to use NVIDIA Containers Getting Started Guide.
  - For non-DGX users, see NVIDIA® GPU Cloud™ (NGC) container registry installation documentation based on your platform.
- Ensure that you have access and can log in to the NGC container registry.

  Refer to NGC Getting Started Guide for more information.

The deep learning frameworks, the NGC Docker containers, and the deep learning framework containers are stored in the `nvcr.io/nvidia` repository.

Chapter 3. Running PyTorch

Before you begin

Before you can run an NGC deep learning framework container, your Docker® environment must support NVIDIA GPUs. To run a container, issue the appropriate command as explained in Running A Container and specify the registry, repository, and tags.

About this task

On a system with GPU support for NGC containers, when you run a container, the following occurs:

- The Docker engine loads the image into a container which runs the software.
- You define the runtime resources of the container by including additional flags and settings that are used with the command.

  These flags and settings are described in Running A Container.
- The GPUs are explicitly defined for the Docker container (defaults to all GPUs, but can be specified by using the `NVIDIA_VISIBLE_DEVICES` environment variable).

  For more information, refer to the nvidia-docker documentation.

Note: Starting in Docker 19.03, complete the steps below.

The method implemented in your system depends on the DGX OS version that you installed (for DGX systems), the NGC Cloud Image that was provided by a Cloud Service Provider, or the software that you installed to prepare to run NGC containers on TITAN PCs, Quadro PCs, or NVIDIA Virtual GPUs (vGPUs).

Procedure

1. Issue the command for the applicable release of the container that you want.

   The following command assumes that you want to pull the latest container.

   ```
   docker pull nvcr.io/nvidia/pytorch:<xx.xx>-py3
   ```
2. Open a command prompt and paste the `pull` command.

Ensure that the pull successfully completes before you proceed to step 3.

3. To run the container image, select one of the following modes:

   - **Interactive**
     - If you have **Docker 19.03 or later**, a typical command to launch the container is:

       ```
       docker run --gpus all -it --rm -v local_dir:container_dir nvcr.io/nvidia/
       pytorch:<xx.xx>-py3
       ```

     - If you have **Docker 19.02 or earlier**, a typical command to launch the container is:

       ```
       nvidia-docker run -it --rm -v local_dir:container_dir nvcr.io/nvidia/
       pytorch:<xx.xx>-py3
       ```

   - **Non-interactive**
     - If you have **Docker 19.03 or later**, a typical command to launch the container is:

       ```
       docker run --gpus all --rm -v local_dir:container_dir nvcr.io/nvidia/
       pytorch:<xx.xx>-py3 <command>
       ```

     - If you have **Docker 19.02 or earlier**, a typical command to launch the container is:

       ```
       nvidia-docker run --rm -v local_dir:container_dir nvcr.io/nvidia/pytorch:<xx.xx>-
       py3 <command>
       ```

   Note: If you use multiprocessing for multi-threaded data loaders, the default shared memory segment size with which the container runs might not be enough. Therefore, you should increase the shared memory size by issuing one of the following commands:

   - `--ipc=host`
   - `--shm-size=<requested memory size>`

   in the command line to

   ```
   docker run --gpus all
   ```

   To pull data and model descriptions from locations outside the container for use by PyTorch or save results to locations outside the container, mount one or more host directories as Docker® data volumes.

Chapter 4. PyTorch Release 26.11

The NVIDIA container image for this PyTorch release is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

The container also includes the following:

- Please refer to CUDA section for the list of libraries inherited from CUDA container.
- NVIDIA CUDA 13.5.0
  - Please refer to the CUDA DL 26.11 release notes section for the list of libraries inherited from the CUDA container.
- Torch-TensorRT 2.15.0a01
- NVIDIA DALI® 2.4.0
- nvImageCodec 0.9.0.20
- MAGMA 2.6.1
- JupyterLab 4.6.6
- TensorRT Model Optimizer 0.47.1
- TransformerEngine 2.21
- NVIDIA cuBLASMp 0.11.0

Driver Requirements

Release 26.11 is based on CUDA 13.5.0. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compatibility information between CUDA versions and driver releases.
- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices.
- NVIDIA Drivers Download - Latest NVIDIA drivers.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 26.11 is based on torch-2.15.0a0+fd23299d96
- In 26.11, the standalone MSLK package is now included for torchao NVFP4 and MXFP quantization paths.
- In 26.09, FlashAttention 4 beta (4.0.0b24) was added, providing CuTeDSL attention kernels alongside the existing FlashAttention 2 package. Additionally the QuACK CuTeDSL kernel library and supporting dependencies are installed.
- The GCP NCCL plugin package is now installed in the container. On Google Cloud Platform hosts (A3 Ultra, A4, A4X and A4X Max machine types), the container automatically detects and activates the GCP NCCL network, tuner, profiler, and environment plugins for optimized multi-node communication.
  - On GCP hosts with the GCP NCCL plugins installed, NCCL telemetry is enabled by default starting in this release.
  - A runtime advisory message is displayed when running on GCP hosts with the GCP NCCL plugins active, directing users to the GCP NCCL telemetry documentation.
- In 26.05, cuSOLVERMp added to the container to support distributed linear algebra solvers.
- The 26.04 release includes BOLT optimizations leveraging LLVM's post-link optimizer to improve CPU code layout and runtime performance.
- PyTorch container image version 26.043 is based on 2.12.0a0+0291f960b62.11.0a0+a6c236b9fd1
- torch.cuda.MemPool() API. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- Starting in 26.11, the DLFW containers introduce general availability support for the NVIDIA Rubin GPU architecture.
- Starting with the 26.08 release, the Apex `fast_multihead_attn`, `fmha`, and `gpu_direct_storage` extensions are no longer built into the PyTorch container.

- Starting with the 26.07 release, the container includes the GCP NCCL (gIB) plugin.
- Starting with the 26.03 release NVIDIA will no longer produce the standalone iGPU containers.
- The 26.05 release will be the last container with the nvfuser library.
- The 26.04 release will be the last container with the lightning-thunder library.
- Effective with the 25.09 release, the RAPIDS libraries are no longer pre-installed in the container. To add RAPIDS to your environment, please follow the official RAPIDS Installation Guide.
- The 25.08 release is the last container with the RAPIDS suite of libraries.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install

a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.

- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
26.11	24.04	NVIDIA CUDA 13.5.0.25	2.15.0a0+fd23299d9e	TensorRT™ 11.5.0.46
26.10		NVIDIA CUDA 13.4.3.001	2.15.0a0+875d8155d2	TensorRT™ 11.4.0.106
26.09		NVIDIA CUDA 13.4.1.	2.14.0a0+b2c75dd0e2	TensorRT™ 11.3
26.08		NVIDIA CUDA 13.4.1.	2.14.0a0+4fdf77b940	TensorRT™ 11.2.1.2
26.07		NVIDIA CUDA 13.3.1.	2.13.0a0+9186a08bae	TensorRT™ 11.1.0.106
26.06		NVIDIA CUDA 13.3.0.	2.13.0a0+8145d630e8	TensorRT™ 11.0.0.114
26.05		NVIDIA CUDA 13.2.1.009	2.12.0a0+5aff3928de	TensorRT™ 10.16.1.11
26.04		NVIDIA CUDA 13.2.1.009	2.12.0a0+0291f960be	TensorRT™ 10.16.1.11
26.03		NVIDIA CUDA 13.2.0.046	2.11.0a0+a6c236b9fd	TensorRT™ 10.16.0.72
26.02		NVIDIA CUDA 13.1.1.006	2.11.0a0+eb65b369f4	TensorRT™ 10.15.1.26
26.01		NVIDIA CUDA 13.1.1.006	2.10.0a0+a36e1d39eb	TensorRT™ 10.14.1.48
25.12		NVIDIA CUDA 13.1.0.36	2.10.0a0+b4e4ee81d3	TensorRT™ 10.14.1.48
25.11		NVIDIA CUDA 13.0.2.006	2.10.0a0+b558c986e8	TensorRT™ 10.14.1.48

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.10		NVIDIA CUDA 13.0.2.006	2.9.0a0+145a3a7bda	TensorRT™ 10.13.3.9
25.09		NVIDIA CUDA 13.0.1.012	2.9.0a0+50eac811a6	TensorRT™ 10.13.3.9
25.08		NVIDIA CUDA 13.0.0.044	2.8.0a0+34c6371d24	TensorRT™ 10.13.2.2
25.06		NVIDIA CUDA 12.9.1	2.8.0a0+5228986c39	TensorRT 10.11.0.33
25.05		NVIDIA CUDA 12.9.0	2.8.0a0+5228986c39	TensorRT 10.10.0.31
25.04		NVIDIA CUDA 12.9.0	2.7.0a0+79aa17489c	TensorRT 10.9.0.34
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a58431	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.
```

```
# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
```

```
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- Transformer Engine version 2.14 shipped with 26.04 has an issue resulting in a nondeterministic wrong answer when using MXFP8 training and bias is present. Please upgrade to Transformer Engine version 2.14.1, where the issue was fixed.
- Flex Attention Kernels may encounter non-deterministic illegal memory access issues on THOR and H100/H200 platforms.
- NVSHMEM shipped with 25.10 and later might require setting `NVIDIA_IMEX_CHANNELS=0` while launching the container (using the docker) to avoid segmentation fault issues.

Chapter 5. PyTorch Release 26.10

The NVIDIA container image for this PyTorch release is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

The container also includes the following:

- Please refer to CUDA section for the list of libraries inherited from CUDA container.
- NVIDIA CUDA 13.4.3
  - Please refer to the CUDA DL 26.10 release notes section for the list of libraries inherited from the CUDA container.
- Torch-TensorRT 2.15.0a01
- NVIDIA DALI® 2.3.0
- nvImageCodec 0.9.0.20
- MAGMA 2.6.1
- JupyterLab 4.6.4
- TensorRT Model Optimizer 0.47.0
- TransformerEngine 2.20
- NVIDIA cuBLASMp 0.10.0

Driver Requirements

Release 26.10 is based on CUDA 13.4.3. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compatibility information between CUDA versions and driver releases.
- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices.
- NVIDIA Drivers Download - Latest NVIDIA drivers.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 26.09 is based on torch-2.15.0a0+875d815502
- In 26.10, FlashAttention 4 beta (4.0.0b24) was added, providing CuTeDSL attention kernels alongside the existing FlashAttention 2 package. Additionally the QuACK CuTeDSL kernel library and supporting dependencies are installed.
- The GCP NCCL plugin package is now installed in the container. On Google Cloud Platform hosts (A3 Ultra, A4, A4X and A4X Max machine types), the container automatically detects and activates the GCP NCCL network, tuner, profiler, and environment plugins for optimized multi-node communication.
  - On GCP hosts with the GCP NCCL plugins installed, NCCL telemetry is enabled by default starting in this release.
  - A runtime advisory message is displayed when running on GCP hosts with the GCP NCCL plugins active, directing users to the GCP NCCL telemetry documentation.
- In 26.05, cuSOLVERMp added to the container to support distributed linear algebra solvers.
- The 26.04 release includes BOLT optimizations leveraging LLVM's post-link optimizer to improve CPU code layout and runtime performance.
- PyTorch container image version 26.043 is based on 2.12.0a0+0291f960b62.11.0a0+a6c236b9fd1
- torch.cuda.MemPool() API. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- The 26.09 DLFW containers introduce initial support for the NVIDIA Rubin GPU architecture.
- Starting with the 26.08 release, the Apex `fast_multihead_attn`, `fmha`, and `gpu_direct_storage` extensions are no longer built into the PyTorch container.
- Starting with the 26.07 release, the container includes the GCP NCCL (gIB) plugin.

- Starting with the 26.03 release NVIDIA will no longer produce the standalone iGPU containers.
- The 26.05 release will be the last container with the nvfuser library.
- The 26.04 release will be the last container with the lightning-thunder library.
- Effective with the 25.09 release, the RAPIDS libraries are no longer pre-installed in the container. To add RAPIDS to your environment, please follow the official RAPIDS Installation Guide.
- The 25.08 release is the last container with the RAPIDS suite of libraries.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example,

using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.

- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
26.10	24.04	NVIDIA CUDA 13.4.3.001	2.15.0a0+875d8155d2	TensorRT™ 11.4.0.106
26.09		NVIDIA CUDA 13.4.1.	2.14.0a0+b2c75dd0d2	TensorRT™ 11.3
26.08		NVIDIA CUDA 13.4.1.	2.14.0a0+4fdf77b940	TensorRT™ 11.2.1.2
26.07		NVIDIA CUDA 13.3.1.	2.13.0a0+9186a08bae	TensorRT™ 11.1.0.106
26.06		NVIDIA CUDA 13.3.0.	2.13.0a0+8145d630e8	TensorRT™ 11.0.0.114
26.05		NVIDIA CUDA 13.2.1.009	2.12.0a0+5aff3928d8	TensorRT™ 10.16.1.11
26.04		NVIDIA CUDA 13.2.1.009	2.12.0a0+0291f960be	TensorRT™ 10.16.1.11
26.03		NVIDIA CUDA 13.2.0.046	2.11.0a0+a6c236b9fd	TensorRT™ 10.16.0.72
26.02		NVIDIA CUDA 13.1.1.006	2.11.0a0+eb65b36914	TensorRT™ 10.15.1.26
26.01		NVIDIA CUDA 13.1.1.006	2.10.0a0+a36e1d39eb	TensorRT™ 10.14.1.48
25.12		NVIDIA CUDA 13.1.0.36	2.10.0a0+b4e4ee81d3	TensorRT™ 10.14.1.48
25.11		NVIDIA CUDA 13.0.2.006	2.10.0a0+b558c986e8	TensorRT™ 10.14.1.48
25.10		NVIDIA CUDA 13.0.2.006	2.9.0a0+145a3a7bda	TensorRT™ 10.13.3.9

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.09		NVIDIA CUDA 13.0.1.012	2.9.0a0+50eac811a6	TensorRT™ 10.13.3.9
25.08		NVIDIA CUDA 13.0.0.044	2.8.0a0+34c6371d24	TensorRT™ 10.13.2.2
25.06		NVIDIA CUDA 12.9.1	2.8.0a0+5228986c39	TensorRT 10.11.0.33
25.05		NVIDIA CUDA 12.9.0	2.8.0a0+5228986c39	TensorRT 10.10.0.31
25.04		NVIDIA CUDA 12.9.0	2.7.0a0+79aa17489c	TensorRT 10.9.0.34
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12			1.4.0a0+174e1ba	TensorRT 6.0.1

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
19.11				
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py
```

```
# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
```

```
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- Transformer Engine version 2.14 shipped with 26.04 has an issue resulting in a nondeterministic wrong answer when using MXFP8 training and bias is present. Please upgrade to Transformer Engine version 2.14.1, where the issue was fixed.
- Flex Attention Kernels may encounter non-deterministic illegal memory access issues on THOR and H100/H200 platforms.
- NVSHMEM shipped with 25.10 and later might require setting `NVIDIA_IMEX_CHANNELS=0` while launching the container (using the docker) to avoid segmentation fault issues.

Chapter 6. PyTorch Release 26.09

The NVIDIA container image for this PyTorch release is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

The container also includes the following:

- Please refer to CUDA section for the list of libraries inherited from CUDA container.
- NVIDIA CUDA 13.4.1
  - Please refer to the CUDA DL 26.09 release notes section for the list of libraries inherited from the CUDA container.
- Torch-TensorRT 2.14
- NVIDIA DALI® 2.3.0
- nvImageCodec 0.9.0
- MAGMA 2.6.1
- JupyterLab 4.6.3
- TensorRT Model Optimizer 0.46
- TransformerEngine 2.19
- NVIDIA cuBLASMp 0.10.0

Driver Requirements

Release 26.09 is based on CUDA 13.4.1. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compatibility information between CUDA versions and driver releases.
- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices.
- NVIDIA Drivers Download - Latest NVIDIA drivers.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 26.09 is based on 2.14.0a0+b2c75dd062
- In 26.09, FlashAttention 4 beta (4.0.0b24) was added, providing CuTeDSL attention kernels alongside the existing FlashAttention 2 package. Additionally the QuACK CuTeDSL kernel library and supporting dependencies are installed.
- The GCP NCCL plugin package is now installed in the container. On Google Cloud Platform hosts (A3 Ultra, A4, A4X and A4X Max machine types), the container automatically detects and activates the GCP NCCL network, tuner, profiler, and environment plugins for optimized multi-node communication.
  - On GCP hosts with the GCP NCCL plugins installed, NCCL telemetry is enabled by default starting in this release.
  - A runtime advisory message is displayed when running on GCP hosts with the GCP NCCL plugins active, directing users to the GCP NCCL telemetry documentation.
- In 26.05, cuSOLVERMp added to the container to support distributed linear algebra solvers.
- The 26.04 release includes BOLT optimizations leveraging LLVM's post-link optimizer to improve CPU code layout and runtime performance.
- PyTorch container image version 26.043 is based on 2.12.0a0+0291f960b62.11.0a0+a6c236b9fd1
- torch.cuda.MemPool() API. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- The 26.09 DLFW containers introduce initial support for the NVIDIA Rubin GPU architecture.
- Starting with the 26.08 release, the Apex `fast_multihead_attn`, `fmha`, and `gpu_direct_storage` extensions are no longer built into the PyTorch container.
- Starting with the 26.07 release, the container includes the GCP NCCL (gIB) plugin.

- Starting with the 26.03 release NVIDIA will no longer produce the standalone iGPU containers.
- The 26.05 release will be the last container with the nvfuser library.
- The 26.04 release will be the last container with the lightning-thunder library.
- Effective with the 25.09 release, the RAPIDS libraries are no longer pre-installed in the container. To add RAPIDS to your environment, please follow the official RAPIDS Installation Guide.
- The 25.08 release is the last container with the RAPIDS suite of libraries.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example,

using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.

- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
26.09	24.04	NVIDIA CUDA 13.4.1.	2.14.0a0+b2c75dd0e2	TensorRT™ 11.3
26.08		NVIDIA CUDA 13.4.1.	2.14.0a0+4fdf77b940	TensorRT™ 11.2.1.2
26.07		NVIDIA CUDA 13.3.1.	2.13.0a0+9186a08bbe	TensorRT™ 11.1.0.106
26.06		NVIDIA CUDA 13.3.0.	2.13.0a0+8145d6306e8	TensorRT™ 11.0.0.114
26.05		NVIDIA CUDA 13.2.1.009	2.12.0a0+5aff3928d8	TensorRT™ 10.16.1.11
26.04		NVIDIA CUDA 13.2.1.009	2.12.0a0+0291f9606e	TensorRT™ 10.16.1.11
26.03		NVIDIA CUDA 13.2.0.046	2.11.0a0+a6c236b9fe1	TensorRT™ 10.16.0.72
26.02		NVIDIA CUDA 13.1.1.006	2.11.0a0+eb65b369f4	TensorRT™ 10.15.1.26
26.01		NVIDIA CUDA 13.1.1.006	2.10.0a0+a36e1d39eb	TensorRT™ 10.14.1.48
25.12		NVIDIA CUDA 13.1.0.36	2.10.0a0+b4e4ee81d3	TensorRT™ 10.14.1.48
25.11		NVIDIA CUDA 13.0.2.006	2.10.0a0+b558c986e8	TensorRT™ 10.14.1.48
25.10		NVIDIA CUDA 13.0.2.006	2.9.0a0+145a3a7bda	TensorRT™ 10.13.3.9
25.09		NVIDIA CUDA 13.0.1.012	2.9.0a0+50eac811a6	TensorRT™ 10.13.3.9

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.08		NVIDIA CUDA 13.0.0.044	2.8.0a0+34c6371d24	TensorRT™ 10.13.2.2
25.06		NVIDIA CUDA 12.9.1	2.8.0a0+5228986c39	TensorRT 10.11.0.33
25.05		NVIDIA CUDA 12.9.0	2.8.0a0+5228986c39	TensorRT 10.10.0.31
25.04		NVIDIA CUDA 12.9.0	2.7.0a0+79aa17489c	TensorRT 10.9.0.34
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	TensorRT 8.0.2.2 for Arm SBSA Linux
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
```

```
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")
```

```
# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- Transformer Engine version 2.14 shipped with 26.04 has an issue resulting in a nondeterministic wrong answer when using MXFP8 training and bias is present. Please upgrade to Transformer Engine version 2.14.1, where the issue was fixed.
- Flex Attention Kernels may encounter non-deterministic illegal memory access issues on THOR and H100/H200 platforms.
- NVSHMEM shipped with 25.10 and later might require setting `NVIDIA_IMEX_CHANNELS=0` while launching the container (using the docker) to avoid segmentation fault issues.

Chapter 7. PyTorch Release 26.08

The NVIDIA container image for this PyTorch release is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

The container also includes the following:

- Please refer to CUDA section for the list of libraries inherited from CUDA container.
- NVIDIA CUDA 13.4.1
  - Please refer to the CUDA DL 26.08 release notes section for the list of libraries inherited from the CUDA container.
- Torch-TensorRT 2.14
- NVIDIA DALI® 2.2.0
- nvImageCodec 0.8.0
- MAGMA 2.6.1
- JupyterLab 4.6.3
- TensorRT Model Optimizer 0.45
- TransformerEngine 2.18
- NVIDIA cuBLASMp 0.10.0

Driver Requirements

Release 26.08 is based on CUDA 13.4.1. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compatibility information between CUDA versions and driver releases.
- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices.
- NVIDIA Drivers Download - Latest NVIDIA drivers.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 26.08 is based on 2.14.0a0+4fdf77b940
- The GCP NCCL plugin package is now installed in the container. On Google Cloud Platform hosts (A3 Ultra, A4, A4X and A4X Max machine types), the container automatically detects and activates the GCP NCCL network, tuner, profiler, and environment plugins for optimized multi-node communication.
  - On GCP hosts with the GCP NCCL plugins installed, NCCL telemetry is enabled by default starting in this release.
  - A runtime advisory message is displayed when running on GCP hosts with the GCP NCCL plugins active, directing users to the GCP NCCL telemetry documentation.
- In 26.05, cuSOLVERMp added to the container to support distributed linear algebra solvers.
- The 26.04 release includes BOLT optimizations leveraging LLVM's post-link optimizer to improve CPU code layout and runtime performance.
- PyTorch container image version 26.043 is based on 2.12.0a0+0291f960b62.11.0a0+a6c236b9fd1
- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- Starting with the 26.08 release, the Apex `fast_multihead_attn`, `fmha`, and `gpu_direct_storage` extensions are no longer built into the PyTorch container.
- Starting with the 26.07 release, the container includes the GCP NCCL (gIB) plugin.
- Starting with the 26.03 release NVIDIA will no longer produce the standalone iGPU containers.
- The 26.05 release will be the last container with the nvfuser library.
- The 26.04 release will be the last container with the lightning-thunder library.

- Effective with the 25.09 release, the RAPIDS libraries are no longer pre-installed in the container. To add RAPIDS to your environment, please follow the official RAPIDS Installation Guide.
- The 25.08 release is the last container with the RAPIDS suite of libraries.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container**.**
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
26.08	24.04	NVIDIA CUDA 13.4.1.	2.14.0a0+4fdf77b940	TensorRT™ 11.2.1.2
26.07		NVIDIA CUDA 13.3.1.	2.13.0a0+9186a08bba	TensorRT™ 11.1.0.106
26.06		NVIDIA CUDA 13.3.0.	2.13.0a0+8145d630e8	TensorRT™ 11.0.0.114
26.05		NVIDIA CUDA 13.2.1.009	2.12.0a0+5aff3928de	TensorRT™ 10.16.1.11
26.04		NVIDIA CUDA 13.2.1.009	2.12.0a0+0291f960be	TensorRT™ 10.16.1.11
26.03		NVIDIA CUDA 13.2.0.046	2.11.0a0+a6c236b9fd	TensorRT™ 10.16.0.72
26.02		NVIDIA CUDA 13.1.1.006	2.11.0a0+eb65b369f4	TensorRT™ 10.15.1.26
26.01		NVIDIA CUDA 13.1.1.006	2.10.0a0+a36e1d39eb	TensorRT™ 10.14.1.48
25.12		NVIDIA CUDA 13.1.0.36	2.10.0a0+b4e4ee81d3	TensorRT™ 10.14.1.48
25.11		NVIDIA CUDA 13.0.2.006	2.10.0a0+b558c986e8	TensorRT™ 10.14.1.48
25.10		NVIDIA CUDA 13.0.2.006	2.9.0a0+145a3a7bda	TensorRT™ 10.13.3.9
25.09		NVIDIA CUDA 13.0.1.012	2.9.0a0+50eac811a6	TensorRT™ 10.13.3.9
25.08		NVIDIA CUDA 13.0.0.044	2.8.0a0+34c6371d24	TensorRT™ 10.13.2.2
25.06		NVIDIA CUDA 12.9.1	2.8.0a0+5228986c39	TensorRT 10.11.0.33
25.05		NVIDIA CUDA 12.9.0	2.8.0a0+5228986c39	TensorRT 10.10.0.31

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.04	22.04	NVIDIA CUDA 12.9.0	2.7.0a0+79aa17489c	TensorRT 10.9.0.34
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10		NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as	TensorRT 5.1.5

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
			well as a cherry-picked	

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
```

```
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- Transformer Engine version 2.14 shipped with 26.04 has an issue resulting in a nondeterministic wrong answer when using MXFP8 training and bias is present. Please upgrade to Transformer Engine version 2.14.1, where the issue was fixed.
- Flex Attention Kernels may encounter non-deterministic illegal memory access issues on THOR and H100/H200 platforms.
- NVSHMEM shipped with 25.10 and later might require setting `NVIDIA_IMEX_CHANNELS=0` while launching the container (using the docker) to avoid segmentation fault issues.

Chapter 8. PyTorch Release 26.07

The NVIDIA container image for this PyTorch release is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

The container also includes the following:

- Please refer to CUDA section for the list of libraries inherited from CUDA container.
- NVIDIA CUDA 13.3.1
  - Please refer to the CUDA DL 26.07 release notes section for the list of libraries inherited from the CUDA container.
- Torch-TensorRT 2.13
- NVIDIA DALI® 2.2.0
- nvImageCodec 0.9.0
- MAGMA 2.6.2
- JupyterLab 4.3.6 including Jupyter-TensorBoard
- TensorRT Model Optimizer 0.44
- TransformerEngine 2.17
- NVIDIA cuBLASMp 0.9.1.3056

Driver Requirements

Release 26.07 is based on CUDA 13.3.1. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compatibility information between CUDA versions and driver releases.
- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices.
- NVIDIA Drivers Download - Latest NVIDIA drivers.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 26.07 is based on 2.13.0a0+9186a08b2c
- The GCP NCCL plugin package is now installed in the container. On Google Cloud Platform hosts (A3 Ultra, A4, A4X and A4X Max machine types), the container automatically detects and activates the GCP NCCL network, tuner, profiler, and environment plugins for optimized multi-node communication.
  - On GCP hosts with the GCP NCCL plugins installed, NCCL telemetry is enabled by default starting in this release.
  - A runtime advisory message is displayed when running on GCP hosts with the GCP NCCL plugins active, directing users to the GCP NCCL telemetry documentation.
- In 26.05, cuSOLVERMp added to the container to support distributed linear algebra solvers.
- The 26.04 release includes BOLT optimizations leveraging LLVM's post-link optimizer to improve CPU code layout and runtime performance.
- PyTorch container image version 26.043 is based on 2.12.0a0+0291f960b62.11.0a0+a6c236b9fd1
- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- Starting with the 26.07 release, the container includes the GCP NCCL (gIB) plugin.
- Starting with the 26.03 release NVIDIA will no longer produce the standalone iGPU containers.
- The 26.05 release will be the last container with the nvfuser library.
- The 26.04 release will be the last container with the lightning-thunder library.
- Effective with the 25.09 release, the RAPIDS libraries are no longer pre-installed in the container. To add RAPIDS to your environment, please follow the official RAPIDS Installation Guide.
- The 25.08 release is the last container with the RAPIDS suite of libraries.

- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
26.07	24.04	NVIDIA CUDA 13.3.1.	2.13.0a0+9186a08bba	TensorRT™ 11.1.0.106
26.06		NVIDIA CUDA 13.3.0.	2.13.0a0+8145d630e8	TensorRT™ 11.0.0.114
26.05		NVIDIA CUDA 13.2.1.009	2.12.0a0+5aff3928de	TensorRT™ 10.16.1.11
26.04		NVIDIA CUDA 13.2.1.009	2.12.0a0+0291f960be	TensorRT™ 10.16.1.11
26.03		NVIDIA CUDA 13.2.0.046	2.11.0a0+a6c236b9fd	TensorRT™ 10.16.0.72
26.02		NVIDIA CUDA 13.1.1.006	2.11.0a0+eb65b3697d	TensorRT™ 10.15.1.26
26.01		NVIDIA CUDA 13.1.1.006	2.10.0a0+a36e1d39eb	TensorRT™ 10.14.1.48
25.12		NVIDIA CUDA 13.1.0.36	2.10.0a0+b4e4ee81d3	TensorRT™ 10.14.1.48
25.11		NVIDIA CUDA 13.0.2.006	2.10.0a0+b558c986e8	TensorRT™ 10.14.1.48
25.10		NVIDIA CUDA 13.0.2.006	2.9.0a0+145a3a7bda	TensorRT™ 10.13.3.9
25.09		NVIDIA CUDA 13.0.1.012	2.9.0a0+50eac811a6	TensorRT™ 10.13.3.9
25.08		NVIDIA CUDA 13.0.0.044	2.8.0a0+34c6371d24	TensorRT™ 10.13.2.2
25.06		NVIDIA CUDA 12.9.1	2.8.0a0+5228986c39	TensorRT 10.11.0.33
25.05		NVIDIA CUDA 12.9.0	2.8.0a0+5228986c39	TensorRT 10.10.0.31
25.04		NVIDIA CUDA 12.9.0	2.7.0a0+79aa17489c	TensorRT 10.9.0.34
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the

overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
```

```
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- Transformer Engine version 2.14 shipped with 26.04 has an issue resulting in a nondeterministic wrong answer when using MXFP8 training and bias is present. Please upgrade to Transformer Engine version 2.14.1, where the issue was fixed.
- Flex Attention Kernels may encounter non-deterministic illegal memory access issues on THOR and H100/H200 platforms.
- NVSHMEM shipped with 25.10 and later might require setting `NVIDIA_IMEX_CHANNELS=0` while launching the container (using the docker) to avoid segmentation fault issues.

Chapter 9. PyTorch Release 26.06

The NVIDIA container image for this PyTorch release is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

The container also includes the following:

- Please refer to CUDA section for the list of libraries inherited from CUDA container.
- NVIDIA CUDA 13.3.0
  - Please refer to the CUDA DL 26.06 release notes section for the list of libraries inherited from the CUDA container.
- Torch-TensorRT 2.13
- NVIDIA DALI® 2.1.1
- nvImageCodec 0.8.0
- MAGMA 2.6.2
- JupyterLab 4.3.6 including Jupyter-TensorBoard
- TensorRT Model Optimizer 0.44
- TransformerEngine 2.16
- NVIDIA cuBLASMp 0.9.0.2693

Driver Requirements

Release 26.06 is based on CUDA 13.3.0. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compatibility information between CUDA versions and driver releases.
- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices.
- NVIDIA Drivers Download - Latest NVIDIA drivers.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 26.06 is based on 2.13.0a0+8145d630e8
- In 26.05, cuSOLVERMp added to the container to support distributed linear algebra solvers.
- The 26.04 release includes BOLT optimizations leveraging LLVM's post-link optimizer to improve CPU code layout and runtime performance.
- PyTorch container image version 26.043 is based on 2.12.0a0+0291f960b62.11.0a0+a6c236b9fd1
- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- Starting with the 26.03 release NVIDIA will no longer produce the standalone iGPU containers.
- The 26.05 release will be the last container with the nvfuser library.
- The 26.04 release will be the last container with the lightning-thunder library.
- Effective with the 25.09 release, the RAPIDS libraries are no longer pre-installed in the container. To add RAPIDS to your environment, please follow the official RAPIDS Installation Guide.
- The 25.08 release is the last container with the RAPIDS suite of libraries.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.

- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
26.06	24.04	NVIDIA CUDA 13.3.0.	2.13.0a0+8145d630e8	TensorRT™ 11.0.0.114
26.05		NVIDIA CUDA 13.2.1.009	2.12.0a0+5aff3928de	TensorRT™ 10.16.1.11

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
26.04		NVIDIA CUDA 13.2.1.009	2.12.0a0+0291f960be	TensorRT™ 10.16.1.11
26.03		NVIDIA CUDA 13.2.0.046	2.11.0a0+a6c236b9fd	TensorRT™ 10.16.0.72
26.02		NVIDIA CUDA 13.1.1.006	2.11.0a0+eb65b369d4	TensorRT™ 10.15.1.26
26.01		NVIDIA CUDA 13.1.1.006	2.10.0a0+a36e1d39eb	TensorRT™ 10.14.1.48
25.12		NVIDIA CUDA 13.1.0.36	2.10.0a0+b4e4ee81d3	TensorRT™ 10.14.1.48
25.11		NVIDIA CUDA 13.0.2.006	2.10.0a0+b558c986e8	TensorRT™ 10.14.1.48
25.10		NVIDIA CUDA 13.0.2.006	2.9.0a0+145a3a7bda	TensorRT™ 10.13.3.9
25.09		NVIDIA CUDA 13.0.1.012	2.9.0a0+50eac811a6	TensorRT™ 10.13.3.9
25.08		NVIDIA CUDA 13.0.0.044	2.8.0a0+34c6371d24	TensorRT™ 10.13.2.2
25.06		NVIDIA CUDA 12.9.1	2.8.0a0+5228986c39	TensorRT 10.11.0.33
25.05		NVIDIA CUDA 12.9.0	2.8.0a0+5228986c39	TensorRT 10.10.0.31
25.04		NVIDIA CUDA 12.9.0	2.7.0a0+79aa17489c	TensorRT 10.9.0.34
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
```

```
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- Transformer Engine version 2.14 shipped with 26.04 has an issue resulting in a nondeterministic wrong answer when using MXFP8 training and bias is present. Please upgrade to Transformer Engine version 2.14.1, where the issue was fixed.
- Flex Attention Kernels may encounter non-deterministic illegal memory access issues on THOR and H100/H200 platforms.
- NVSHMEM shipped with 25.10 and later might require setting `NVIDIA_IMEX_CHANNELS=0` while launching the container (using the docker) to avoid segmentation fault issues.

Chapter 10. PyTorch Release 26.05

The NVIDIA container image for this PyTorch release is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

The container also includes the following:

- Please refer to CUDA section for the list of libraries inherited from CUDA container.
- NVIDIA CUDA 13.2.1.009
  - Please refer to the CUDA DL 26.05 release notes section for the list of libraries inherited from the CUDA container.
- Torch-TensorRT 2.12.0a0
- NVIDIA DALI® 2.1.0
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2
- JupyterLab 4.3.6 including Jupyter-TensorBoard
- TensorRT Model Optimizer 0.43
- TransformerEngine 2.15
- NVIDIA cuBLASMp0.8.1.2.360

Driver Requirements

Release 26.05 is based on CUDA 13.2.1. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compatibility information between CUDA versions and driver releases.
- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices.
- NVIDIA Drivers Download - Latest NVIDIA drivers.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 26.05 is based on 2.12.0a0+5aff3928d8
- In 26.05, cuSOLVERMp added to the container to support distributed linear algebra solvers.
- The 26.04 release includes BOLT optimizations leveraging LLVM's post-link optimizer to improve CPU code layout and runtime performance.
- PyTorch container image version 26.043 is based on 2.12.0a0+0291f960b62.11.0a0+a6c236b9fd1
- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- Starting with the 26.03 release NVIDIA will no longer produce the standalone iGPU containers.
- The 26.05 release will be the last container with the nvfuser library.
- The 26.04 release will be the last container with the lightning-thunder library.
- Effective with the 25.09 release, the RAPIDS libraries are no longer pre-installed in the container. To add RAPIDS to your environment, please follow the official RAPIDS Installation Guide.
- The 25.08 release is the last container with the RAPIDS suite of libraries.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.

- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
26.05	24.04	NVIDIA CUDA 13.2.1.009	2.12.0a0+5aff3928de	TensorRT™ 10.16.1.11
26.04		NVIDIA CUDA 13.2.1.009	2.12.0a0+0291f9601e	TensorRT™ 10.16.1.11

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
26.03		NVIDIA CUDA 13.2.0.046	2.11.0a0+a6c236b9fd	TensorRT™ 10.16.0.72
26.02		NVIDIA CUDA 13.1.1.006	2.11.0a0+eb65b36914	TensorRT™ 10.15.1.26
26.01		NVIDIA CUDA 13.1.1.006	2.10.0a0+a36e1d39eb	TensorRT™ 10.14.1.48
25.12		NVIDIA CUDA 13.1.0.36	2.10.0a0+b4e4ee81d3	TensorRT™ 10.14.1.48
25.11		NVIDIA CUDA 13.0.2.006	2.10.0a0+b558c986e8	TensorRT™ 10.14.1.48
25.10		NVIDIA CUDA 13.0.2.006	2.9.0a0+145a3a7bda	TensorRT™ 10.13.3.9
25.09		NVIDIA CUDA 13.0.1.012	2.9.0a0+50eac811a6	TensorRT™ 10.13.3.9
25.08		NVIDIA CUDA 13.0.0.044	2.8.0a0+34c6371d24	TensorRT™ 10.13.2.2
25.06		NVIDIA CUDA 12.9.1	2.8.0a0+5228986c39	TensorRT 10.11.0.33
25.05		NVIDIA CUDA 12.9.0	2.8.0a0+5228986c39	TensorRT 10.10.0.31
25.04		NVIDIA CUDA 12.9.0	2.7.0a0+79aa17489c	TensorRT 10.9.0.34
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.

- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)
```

```
allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- Transformer Engine version 2.14 shipped with 26.04 has an issue resulting in a nondeterministic wrong answer when using MXFP8 training and bias is present. Please upgrade to Transformer Engine version 2.14.1, where the issue was fixed.
- Flex Attention Kernels may encounter non-deterministic illegal memory access issues on THOR and H100/H200 platforms.
- NVSHMEM shipped with 25.10 and later might require setting `NVIDIA_IMEX_CHANNELS=0` while launching the container (using the docker) to avoid segmentation fault issues.

Chapter 11. PyTorch Release 26.04

The NVIDIA container image for this PyTorch release is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

The container also includes the following:

- Please refer to CUDA section for the list of libraries inherited from CUDA container.
- NVIDIA CUDA 13.2.1.009
  - Please refer to the CUDA DL 26.04 release notes section for the list of libraries inherited from the CUDA container.
- Torch-TensorRT 2.12.0a0
- NVIDIA DALI® 2.0.0
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2
- JupyterLab 4.3.6 including Jupyter-TensorBoard
- TensorRT Model Optimizer 0.42
- TransformerEngine 2.14
- NVIDIA cuBLASMp0.8.1.2.360

Driver Requirements

Release 26.04 is based on CUDA 13.2.1. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compatibility information between CUDA versions and driver releases.
- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices.
- NVIDIA Drivers Download - Latest NVIDIA drivers.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 26.04 is based on 2.12.0a0+0291f960b62.11.0a0+a6c236b9fd1.
- The 26.04 release includes BOLT optimizations leveraging LLVM's post-link optimizer to improve CPU code layout and runtime performance. For more details see: https://github.com/llvm/llvm-project/blob/main/bolt/README.md.
- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- Starting with the 26.03 release NVIDIA will no longer produce the standalone iGPU containers.
- The 26.05 release will be the last container with the nvfuser library.
- The 26.04 release will be the last container with the lightning-thunder library.
- Effective with the 25.09 release, the RAPIDS libraries are no longer pre-installed in the container. To add RAPIDS to your environment, please follow the official RAPIDS Installation Guide.
- The 25.08 release is the last container with the RAPIDS suite of libraries.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.

- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
26.04	24.04	NVIDIA CUDA 13.2.1.009	2.12.0a0+0291f960be	TensorRT™ 10.16.1.11
26.03		NVIDIA CUDA 13.2.0.046	2.11.0a0+a6c236b9fd	TensorRT™ 10.16.0.72
26.02		NVIDIA CUDA 13.1.1.006	2.11.0a0+eb65b36914	TensorRT™ 10.15.1.26
26.01		NVIDIA CUDA 13.1.1.006	2.10.0a0+a36e1d39eb	TensorRT™ 10.14.1.48

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.12		NVIDIA CUDA 13.1.0.36	2.10.0a0+b4e4ee81d3	TensorRT™ 10.14.1.48
25.11		NVIDIA CUDA 13.0.2.006	2.10.0a0+b558c986e8	TensorRT™ 10.14.1.48
25.10		NVIDIA CUDA 13.0.2.006	2.9.0a0+145a3a7bda	TensorRT™ 10.13.3.9
25.09		NVIDIA CUDA 13.0.1.012	2.9.0a0+50eac811a6	TensorRT™ 10.13.3.9
25.08		NVIDIA CUDA 13.0.0.044	2.8.0a0+34c6371d24	TensorRT™ 10.13.2.2
25.06		NVIDIA CUDA 12.9.1	2.8.0a0+5228986c39	TensorRT 10.11.0.33
25.05		NVIDIA CUDA 12.9.0	2.8.0a0+5228986c39	TensorRT 10.10.0.31
25.04		NVIDIA CUDA 12.9.0	2.7.0a0+79aa17489c	TensorRT 10.9.0.34
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions

for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
```

```
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- Transformer Engine version 2.14 shipped with 26.04 has an issue resulting in a nondeterministic wrong answer when using MXFP8 training and bias is present. Please upgrade to Transformer Engine version 2.14.1, where the issue was fixed.
- Flex Attention Kernels may encounter non-deterministic illegal memory access issues on THOR and H100/H200 platforms.
- NVSHMEM shipped with 25.10 and later might require setting `NVIDIA_IMEX_CHANNELS=0` while launching the container (using the docker) to avoid segmentation fault issues.

Chapter 12. PyTorch Release 26.03

The NVIDIA container image for this PyTorch release is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

The container also includes the following:

- Please refer to CUDA section for the list of libraries inherited from CUDA container.
- NVIDIA CUDA 13.2.0.046
  - Please refer to the CUDA DL 26.03 release notes section for the list of libraries inherited from the CUDA container.
- Torch-TensorRT 2.11.0a0
- NVIDIA DALI® 2.0
- nvImageCodec 0.2.0.7
- DCGM 4.4.2-1
- MAGMA 2.6.2
- JupyterLab 4.3.6 including Jupyter-TensorBoard
- TensorRT Model Optimizer 0.41
- TransformerEngine 2.13
- NVIDIA cuBLASMp0.8.0.2023

Driver Requirements

Release 26.03 is based on CUDA 13.2.0. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compatibility information between CUDA versions and driver releases.
- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices.
- NVIDIA Drivers Download - Latest NVIDIA drivers.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 26.03 is based on 2.11.0a0+a6c236b9fd1.
- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- Starting with the 26.03 release NVIDIA will no longer produce the standalone iGPU containers.
- The 26.04 release will be the last container with the lightning-thunder library.
- Effective with the 25.09 release, the RAPIDS libraries are no longer pre-installed in the container. To add RAPIDS to your environment, please follow the official RAPIDS Installation Guide.
- The 25.08 release is the last container with the RAPIDS suite of libraries.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer

to the Frameworks Support Matrix for information regarding which iGPU hardware/ software is supported by which container.

- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
26.03	24.04	NVIDIA CUDA 13.2.0.046	2.11.0a0+a6c236b9fd	TensorRT™ 10.16.0.72
26.02		NVIDIA CUDA 13.1.1.006	2.11.0a0+eb65b36914	TensorRT™ 10.15.1.26
26.01		NVIDIA CUDA 13.1.1.006	2.10.0a0+a36e1d39eb	TensorRT™ 10.14.1.48
25.12		NVIDIA CUDA 13.1.0.36	2.10.0a0+b4e4ee81d3	TensorRT™ 10.14.1.48
25.11		NVIDIA CUDA 13.0.2.006	2.10.0a0+b558c986e8	TensorRT™ 10.14.1.48
25.10		NVIDIA CUDA 13.0.2.006	2.9.0a0+145a3a7bda	TensorRT™ 10.13.3.9

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.09		NVIDIA CUDA 13.0.1.012	2.9.0a0+50eac811a6	TensorRT™ 10.13.3.9
25.08		NVIDIA CUDA 13.0.0.044	2.8.0a0+34c6371d24	TensorRT™ 10.13.2.2
25.06		NVIDIA CUDA 12.9.1	2.8.0a0+5228986c39	TensorRT 10.11.0.33
25.05		NVIDIA CUDA 12.9.0	2.8.0a0+5228986c39	TensorRT 10.10.0.31
25.04		NVIDIA CUDA 12.9.0	2.7.0a0+79aa17489c	TensorRT 10.9.0.34
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12			1.4.0a0+174e1ba	TensorRT 6.0.1

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
19.11				
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py
```

```
# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
```

```
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- Flex Attention Kernels may encounter non-deterministic illegal memory access issues on THOR and H100/H200 platforms.
- NVSHMEM shipped with 25.10 and later might require setting `NVIDIA_IMEX_CHANNELS=0` while launching the container (using the docker) to avoid segmentation fault issues.

Chapter 13. PyTorch Release 26.02

The NVIDIA container image for this PyTorch release is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

The container also includes the following:

- Please refer to CUDA section for the list of libraries inherited from CUDA container.
- NVIDIA CUDA 13.1.1
  - Please refer to the CUDA DL 26.02 release notes section for the list of libraries inherited from the CUDA container.
- Torch-TensorRT 2.11.0a0
- NVIDIA DALI® 1.53
- nvImageCodec 0.2.0.7
- ONNX Runtime 1.23.2
- Intel OpenVINO 2025.4.0
- DCGM 4.4.2-1
- MAGMA 2.6.2
- JupyterLab 4.3.6 including Jupyter-TensorBoard
- TensorRT Model Optimizer 0.41
- TransformerEngine 2.12
- NVIDIA cuBLASMp 0.7.0.125

Driver Requirements

Release 26.02 is based on CUDA 13.1.1. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compatibility information between CUDA versions and driver releases.

- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices.
- NVIDIA Drivers Download - Latest NVIDIA drivers.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 26.02 is based on 2.11.0a0+eb65b36914.
- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- Starting with the 26.03 release NVIDIA will no longer produce the standalone iGPU containers.
- The 26.04 release will be the last container with the lightning-thunder library.
- Effective with the 25.09 release, the RAPIDS libraries are no longer pre-installed in the container. To add RAPIDS to your environment, please follow the official RAPIDS Installation Guide.
- The 25.08 release is the last container with the RAPIDS suite of libraries.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).

- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
26.02	24.04	NVIDIA CUDA 13.1.1.006	2.11.0a0+eb65b369	TensorRT™ 10.15.1.26
26.01		NVIDIA CUDA 13.1.1.006	2.10.0a0+a36e1d39	TensorRT™ 10.14.1.48
25.12		NVIDIA CUDA 13.1.0.36	2.10.0a0+b4e4ee81	TensorRT™ 10.14.1.48
25.11		NVIDIA CUDA 13.0.2.006	2.10.0a0+b558c986	TensorRT™ 10.14.1.48
25.10		NVIDIA CUDA 13.0.2.006	2.9.0a0+145a3a7bda	TensorRT™ 10.13.3.9

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.09		NVIDIA CUDA 13.0.1.012	2.9.0a0+50eac811a6	TensorRT™ 10.13.3.9
25.08		NVIDIA CUDA 13.0.0.044	2.8.0a0+34c6371d24	TensorRT™ 10.13.2.2
25.06		NVIDIA CUDA 12.9.1	2.8.0a0+5228986c39	TensorRT 10.11.0.33
25.05		NVIDIA CUDA 12.9.0	2.8.0a0+5228986c39	TensorRT 10.10.0.31
25.04		NVIDIA CUDA 12.9.0	2.7.0a0+79aa17489c	TensorRT 10.9.0.34
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12			1.4.0a0+174e1ba	TensorRT 6.0.1

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
19.11				
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py
```

```
# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
```

```
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- Flex Attention Kernels may encounter non-deterministic illegal memory access issues on THOR and H100/H200 platforms.
- An LLVM/Triton v3.5 update exposed a PTXAS bug that causes numerical issues for torch.compile and maxpool_2d op. See this issue for more details. Update to Triton to v3.6 might resolve this issue.
- NVSHMEM shipped with 25.10 and later might require setting `NVIDIA_IMEX_CHANNELS=0` while launching the container (using the docker) to avoid segmentation fault issues.
- Running pytorch numba applications with upstream/vanilla numba on driver 580.82.07 based systems may encounter segmentation fault, we have applied a workaround in the container, please do not overwrite numba. See:https://github.com/pytorch/pytorch/issues/162878

Chapter 14. PyTorch Release 26.01

The NVIDIA container image for this PyTorch release is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

The container also includes the following:

- Please refer to CUDA section for the list of libraries inherited from CUDA container.
- NVIDIA CUDA 13.1.1
  - Please refer to the CUDA DL 26.01 release notes section for the list of libraries inherited from the CUDA container.
- Torch-TensorRT 2.10.0a0
- NVIDIA DALI® 1.53
- nvImageCodec 0.2.0.7
- ONNX Runtime 1.23.2
- Intel OpenVINO 2025.4.0
- DCGM 4.4.2-1
- MAGMA 2.6.2
- JupyterLab 4.3.6 including Jupyter-TensorBoard
- TensorRT Model Optimizer 0.39
- TransformerEngine 2.11
- NVIDIA cuBLASMp 0.7.0.125

Driver Requirements

Release 26.01 is based on CUDA 13.1.1. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compatibility information between CUDA versions and driver releases.

- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices.
- NVIDIA Drivers Download - Latest NVIDIA drivers.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 26.01 is based on 2.10.0a0+a36e1d39eb.
- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- The 26.04 release will be the last container with the lightning-thunder library.
- Effective with the 25.09 release, the RAPIDS libraries are no longer pre-installed in the container. To add RAPIDS to your environment, please follow the official RAPIDS Installation Guide.
- The 25.08 release is the last container with the RAPIDS suite of libraries.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer

to the Frameworks Support Matrix for information regarding which iGPU hardware/ software is supported by which container.

- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
26.01	24.04	NVIDIA CUDA 13.1.1.006	2.10.0a0+a36e1d39eb	TensorRT™ 10.14.1.48
25.12		NVIDIA CUDA 13.1.0.36	2.10.0a0+b4e4ee81d3	TensorRT™ 10.14.1.48
25.11		NVIDIA CUDA 13.0.2.006	2.10.0a0+b558c986e8	TensorRT™ 10.14.1.48
25.10		NVIDIA CUDA 13.0.2.006	2.9.0a0+145a3a7bda	TensorRT™ 10.13.3.9
25.09		NVIDIA CUDA 13.0.1.012	2.9.0a0+50eac811a6	TensorRT™ 10.13.3.9
25.08		NVIDIA CUDA 13.0.0.044	2.8.0a0+34c6371d24	TensorRT™ 10.13.2.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.06		NVIDIA CUDA 12.9.1	2.8.0a0+5228986c39	TensorRT 10.11.0.33
25.05		NVIDIA CUDA 12.9.0	2.8.0a0+5228986c39	TensorRT 10.10.0.31
25.04		NVIDIA CUDA 12.9.0	2.7.0a0+79aa17489c	TensorRT 10.9.0.34
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream	TensorRT 5.1.5

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
			commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
```

```
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])
```

```
# release memory to system
del tensor, del pool
```

Known Issues

- Flex Attention Kernels may encounter non-deterministic illegal memory access issues on THOR and H100/H200 platforms.
- An LLVM/Triton v3.5 update exposed a PTXAS bug that causes numerical issues for torch.compile and maxpool_2d op. See this issue for more details. Update to Triton to v3.6 might resolve this issue.
- NVSHMEM shipped with 25.10 and later might require setting `NVIDIA_IMEX_CHANNELS=0` while launching the container (using the docker) to avoid segmentation fault issues.
- Running pytorch numba applications with upstream/vanilla numba on driver 580.82.07 based systems may encounter segmentation fault, we have applied a workaround in the container, please do not overwrite numba. See:https://github.com/pytorch/pytorch/issues/162878

Chapter 15. PyTorch Release 25.12

The NVIDIA container image for this PyTorch release is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

The container also includes the following:

- Please refer to CUDA section for the list of libraries inherited from CUDA container.
- NVIDIA CUDA 13.1.0
  - Please refer to the CUDA DL 25.12 release notes section for the list of libraries inherited from the CUDA container.
- Torch-TensorRT 2.10.0a0
- NVIDIA DALI® 1.52
- nvImageCodec 0.2.0.7
- ONNX Runtime 1.23.2
- Intel OpenVINO 2025.4.0
- DCGM 4.4.2-1
- MAGMA 2.6.2
- JupyterLab 4.3.6 including Jupyter-TensorBoard
- TensorRT Model Optimizer 0.39
- TransformerEngine 2.10
- NVIDIA cuBLASMp 0.7.0.125

Driver Requirements

Release 25.12 is based on CUDA 13.1.0. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compatibility information between CUDA versions and driver releases.

- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices.
- NVIDIA Drivers Download - Latest NVIDIA drivers.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 25.12 is based on 2.10.0a0+b4e4ee81d3.
- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- Effective with the 25.09 release, the RAPIDS libraries are no longer pre-installed in the container. To add RAPIDS to your environment, please follow the official RAPIDS Installation Guide.
- The 25.08 release is the last container with the RAPIDS suite of libraries.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.

- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.12	24.04	NVIDIA CUDA 13.1.0.36	2.10.0a0+b4e4ee81d3	TensorRT™ 10.14.1.48
25.11		NVIDIA CUDA 13.0.2.006	2.10.0a0+b558c986e8	TensorRT™ 10.14.1.48
25.10		NVIDIA CUDA 13.0.2.006	2.9.0a0+145a3a7bda	TensorRT™ 10.13.3.9
25.09		NVIDIA CUDA 13.0.1.012	2.9.0a0+50eac811a6	TensorRT™ 10.13.3.9
25.08		NVIDIA CUDA 13.0.0.044	2.8.0a0+34c6371d24	TensorRT™ 10.13.2.2
25.06		NVIDIA CUDA 12.9.1	2.8.0a0+5228986c39	TensorRT 10.11.0.33
25.05		NVIDIA CUDA 12.9.0	2.8.0a0+5228986c39	TensorRT 10.10.0.31

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.04		NVIDIA CUDA 12.9.0	2.7.0a0+79aa17489c	TensorRT 10.9.0.34
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as	TensorRT 5.1.5

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
			well as a cherry-picked	

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
```

```
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- Flex Attention Kernels may encounter non-deterministic illegal memory access issues on THOR and H100/H200 platforms.
- An LLVM/Triton v3.5 update exposed a PTXAS bug that causes numerical issues for torch.compile and maxpool_2d op. See this issue for more details. Update Triton to v3.6 might resolve this issue.
- NVSHMEM shipped with 25.10 and later might require setting `NVIDIA_IMEX_CHANNELS=0` while launching the container (using the docker) to avoid segmentation fault issues.
- Running pytorch numba applications with upstream/vanilla numba on driver 580.82.07 based systems may encounter segmentation fault, we have applied a workaround in the container, please do not overwrite numba. See:https://github.com/pytorch/pytorch/issues/162878

Chapter 16. PyTorch Release 25.11

The NVIDIA container image for this PyTorch release is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

The container also includes the following:

- Please refer to CUDA section for the list of libraries inherited from CUDA container.
- NVIDIA CUDA 13.0.2.006
- Torch-TensorRT 2.10.0a0
- NVIDIA DALI® 1.52
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2
- JupyterLab 4.3.6 including Jupyter-TensorBoard
- TensorRT Model Optimizer 0.37
- TransformerEngine 2.9
- NVIDIA cuBLASMp 0.6.0.84

Driver Requirements

Release 25.11 is based on CUDA 13.0.2. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compatibility information between CUDA versions and driver releases.
- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices.
- NVIDIA Drivers Download - Latest NVIDIA drivers.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 25.11 is based on 2.10.0a0+b558c986e8.
- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- Effective with the 25.09 release, the RAPIDS libraries are no longer pre-installed in the container. To add RAPIDS to your environment, please follow the official RAPIDS Installation Guide.
- The 25.08 release is the last container with the RAPIDS suite of libraries.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization.

Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.11	24.04	NVIDIA CUDA 13.0.2.006	2.10.0a0+b558c986e8	TensorRT™ 10.14.1.48
25.10		NVIDIA CUDA 13.0.2.006	2.9.0a0+145a3a7bda	TensorRT™ 10.13.3.9
25.09		NVIDIA CUDA 13.0.1.012	2.9.0a0+50eac811a6	TensorRT™ 10.13.3.9
25.08		NVIDIA CUDA 13.0.0.044	2.8.0a0+34c6371d24	TensorRT™ 10.13.2.2
25.06		NVIDIA CUDA 12.9.1	2.8.0a0+5228986c39	TensorRT 10.11.0.33
25.05		NVIDIA CUDA 12.9.0	2.8.0a0+5228986c39	TensorRT 10.10.0.31
25.04		NVIDIA CUDA 12.9.0	2.7.0a0+79aa17489c	TensorRT 10.9.0.34
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision

training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}
```

```
void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- An LLVM/Triton v3.5 update exposed a PTXAS bug that causes numerical issues for torch.compile and maxpool_2d op. See this issue for more details.
- NVSHMEM shipped with 25.10 and later might require setting `NVIDIA_IMEX_CHANNELS=0` while launching the container (using the docker) to avoid segmentation fault issues.
- Running pytorch numba applications with upstream/vanilla numba on driver 580.82.07 based systems may encounter segmentation fault, we have applied a workaround

in the container, please do not overwrite numba. See:https://github.com/pytorch/pytorch/issues/162878

Chapter 17. PyTorch Release 25.10

The NVIDIA container image for this PyTorch release is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

The container also includes the following:

- Please refer to CUDA section for the list of libraries inherited from CUDA container.
- Torch-TensorRT 2.9.0a0
- NVIDIA DALI® 1.51.2
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2
- JupyterLab 4.3.6 including Jupyter-TensorBoard
- TensorRT Model Optimizer 0.33
- TransformerEngine 2.8
- NVIDIA cuBLASMp 0.5.1.65

Driver Requirements

Release 25.10 is based on CUDA 13.0.2. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compatibility information between CUDA versions and driver releases.
- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices.
- NVIDIA Drivers Download - Latest NVIDIA drivers.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 25.10 is based on 2.9.0a0+145a3a7bda.

- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- Effective with the 25.09 release, the RAPIDS libraries are no longer pre-installed in the container. To add RAPIDS to your environment, please follow the official RAPIDS Installation Guide.
- The 25.08 release is the last container with the RAPIDS suite of libraries.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular

Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.10	24.04	NVIDIA CUDA 13.0.2.006	2.9.0a0+145a3a7bda	TensorRT™ 10.13.3.9
25.09		NVIDIA CUDA 13.0.1.012	2.9.0a0+50eac811a6	TensorRT™ 10.13.3.9
25.08		NVIDIA CUDA 13.0.0.044	2.8.0a0+34c6371d24	TensorRT™ 10.13.2.2
25.06		NVIDIA CUDA 12.9.1	2.8.0a0+5228986c39	TensorRT 10.11.0.33
25.05		NVIDIA CUDA 12.9.0	2.8.0a0+5228986c39	TensorRT 10.10.0.31
25.04		NVIDIA CUDA 12.9.0	2.7.0a0+79aa17489c	TensorRT 10.9.0.34
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with

FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
```

```
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- NVSHMEM shipped with 25.10 might require setting `NVIDIA_IMEX_CHANNELS=0` while launching the container using the docker to avoid segmentation fault issues.
- Torch_TensorRT does not support `torch.ops.aten.nonzero.default` on Thor, due to limited support in TensorRT 10.13. Models using this operation will fail with the error - "Could not find any implementation for node [trainStation3]. In computeCosts at optimizer/common/tactic/optimizer.cpp:4115".
- Running pytorch numba applications with upstream/vanilla numba on driver 580.82.07 based systems may encounter segmentation fault, we have applied a workaround in the container, please do not overwrite numba. See:https://github.com/pytorch/pytorch/issues/162878

Chapter 18. PyTorch Release 25.09

The NVIDIA container image for this PyTorch release is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

The container also includes the following:

- Please refer to CUDA section for the list of libraries inherited from CUDA container.
- Torch-TensorRT 2.9.0a0
- NVIDIA DALI® 1.51.2
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2
- JupyterLab 4.3.6 including Jupyter-TensorBoard
- TensorRT Model Optimizer 0.33
- TransformerEngine 2.7
- NVIDIA cuBLASMp 0.5.1.65

Driver Requirements

Release 25.09 is based on CUDA 13.0.1. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compatibility information between CUDA versions and driver releases.
- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices.
- NVIDIA Drivers Download - Latest NVIDIA drivers.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 25.09 is based on 2.9.0a0+50eac811a6.

- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- Effective with the 25.09 release, the RAPIDS libraries are no longer pre-installed in the container. To add RAPIDS to your environment, please follow the official RAPIDS Installation Guide.
- The 25.08 release is the last container with the RAPIDS suite of libraries.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular

Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.09	24.04	NVIDIA CUDA 13.0.1.012	2.9.0a0+50eac811a6	TensorRT™ 10.13.3.9
25.08		NVIDIA CUDA 13.0.0.044	2.8.0a0+34c6371d24	TensorRT™ 10.13.2.2
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()
```

```
# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- Torch_TensorRT does not support `torch.ops.aten.nonzero.default on Thor`
- Running pytorch numba applications with upstream/vanilla numba on driver 580.82.07 based systems may encounter segmentation fault, we have applied a workaround in the container, please do not overwrite numba. See: https://github.com/pytorch/pytorch/issues/162878

Chapter 19. PyTorch Release 25.08

The NVIDIA container image for this PyTorch release is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

The container also includes the following:

- Please refer to CUDA section for the list of libraries inherited from CUDA container.
- Torch-TensorRT 2.8.0a0
- NVIDIA DALI® 1.51.2
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2
- JupyterLab 4.3.5 including Jupyter-TensorBoard
- TensorRT Model Optimizer 0.33
- TransformerEngine 2.5
- NVIDIA RAPIDS™ 25.06
- NVIDIA cuBLASMp 0.5.1.65

Driver Requirements

Release 25.08 is based on CUDA 13.0.0. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compatibility information between CUDA versions and driver releases.
- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices.
- NVIDIA Drivers Download - Latest NVIDIA drivers.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 25.08 is based on 2.8.0a0+34c6371d24.
- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- The 25.08 release is the last container with the RAPIDS suite of libraries.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.08	24.04	NVIDIA CUDA 13.0.0.044	2.8.0a0+34c6371d24	TensorRT™ 10.13.2.2
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions

for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
```

```
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- Certain distributed jobs may encounter UCX errors involving MLX5, in which case NVIDIA recommends using the environment variable as workaround: UCX_DC_MLX5_DDP_ENABLE=n.

Chapter 20. PyTorch Release 25.06

The NVIDIA container image for PyTorch, release 25.06 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

Please refer to CUDA release notes for the list of libraries inherited from CUDA container.

- CUDA 12.9.1
  - For the full set of CUDA libraries, please refer to CUDA DL 25.06 Release Notes
- Torch-TensorRT 2.8.0a0
- NVIDIA DALI® 1.50
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2
- JupyterLab 4.3.6 including Jupyter-TensorBoard
- TensorRT Model Optimizer 0.29
- TransformerEngine 2.4
- NVIDIA RAPIDS™ 25.04
- NVIDIA cuBLASMp 0.4.0

Driver Requirements

Release 25.06 is based on CUDA 12.9.1. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compati.bility information between CUDA versions and driver releases.
- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices.
- NVIDIA Drivers Download - Latest NVIDIA drivers.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 25.06 is based on 2.8.0a0+5228986c39.
- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular

Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.06	24.04	NVIDIA CUDA 12.9.1	2.8.0a0+5228986c39	TensorRT 10.11.0.33
25.05		NVIDIA CUDA 12.9.0	2.8.0a0+5228986c39	TensorRT 10.10.0.31
25.04		NVIDIA CUDA 12.9.0	2.7.0a0+79aa17489c	TensorRT 10.9.0.34
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a58436	TensorRT 10.4.0.26

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.

- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)
```

```
allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- None.

Chapter 21. PyTorch Release 25.05

The NVIDIA container image for PyTorch, release 25.05 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

- CUDA 12.9.0
  - For the full set of CUDA libraries, please refer to the CUDA DL 25.05 Release Notes
- Torch-TensorRT 2.8.0a0
- NVIDIA DALI® 1.49
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2
- JupyterLab 4.3.6 including Jupyter-TensorBoard
- TensorRT Model Optimizer 0.25
- TransformerEngine 2.3
- NVIDIA RAPIDS™ 25.04
- NVIDIA cuBLASMp 0.4.0

Driver Requirements

Release 25.05 is based on CUDA 12.9.0. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compatibility information between CUDA versions and driver releases.
- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices
- NVIDIA Drivers Download - Latest NVIDIA drivers

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 25.05 is based on 2.8.0a0+5228986c39.
- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed

and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.

- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.05	24.04	NVIDIA CUDA 12.9.0	2.8.0a0+5228986c39	TensorRT 10.10.0.31
25.04		NVIDIA CUDA 12.9.0	2.7.0a0+79aa17489c	TensorRT 10.9.0.34
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()
```

```
# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- None.

Chapter 22. PyTorch Release 25.04

The NVIDIA container image for PyTorch, release 25.04 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

- CUDA 12.9.0
  - For the full set of CUDA libraries, please refer to the CUDA DL 25.04 Release Notes
- Torch-TensorRT 2.7.0a0
- NVIDIA DALI® 1.48
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2
- JupyterLab 4.3.6 including Jupyter-TensorBoard
- TensorRT Model Optimizer 0.25
- TransformerEngine 2.2
- NVIDIA RAPIDS™ 25.02
- NVIDIA cuBLASMp 0.4.0

Driver Requirements

Release 25.04 is based on CUDA 12.9.0. For comprehensive and up-to-date driver compatibility information, please refer to the following documentation:

- NVIDIA CUDA Compatibility Guide - Compatibility information between CUDA versions and driver releases.
- CUDA Toolkit Release Notes - Driver version requirements and compatibility matrices
- NVIDIA Drivers Download - Latest NVIDIA drivers

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 25.04 is based on 2.7.0a0+79aa17489c.
- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed

and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.

- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.04	24.04	NVIDIA CUDA 12.9.0	2.7.0a0+79aa17489c	TensorRT 10.9.0.34
25.03		NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions

for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
```

```
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- None.

Chapter 23. PyTorch Release 25.03

The NVIDIA container image for PyTorch, release 25.03 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 24.04 including Python 3.12
- NVIDIA CUDA 12.8.1.012
- NVIDIA cuBLAS 12.8.4.1
- NVIDIA cuBLASMp 0.4.0
- NVIDIA cuDNN 9.8.0.87
- NVIDIA NCCL 2.25.1
- NVIDIA RAPIDS™ 25.02
- rdma-core 50.0
- NVIDIA HPC-X 2.21
- SHARP 3.9.0
- UCC 1.4.0
- UCX 1.18.0
- OpenMPI 4.1.7
- GDRCopy 2.4.1
- TensorBoard 2.16.2
- Nsight Compute 2025.1.1.2
- Nsight Systems 2025.1.1.110
- NVIDIA TensorRT™ 10.9.0.34
- Torch-TensorRT 2.7.0a0
- NVIDIA DALI® 1.47
- nvImageCodec 0.2.0.7

- MAGMA 2.6.2
- JupyterLab 4.3.5 including Jupyter-TensorBoard
- TransformerEngine 2.1
- TensorRT Model Optimizer 0.23

Driver Requirements

Release 25.03 is based on CUDA 12.8.1.012 which requires NVIDIA Driver release 570 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 470.57 (or later R470), 525.85 (or later R525), 535.86 (or later R535), or 545.23 (or later R545).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R450, R460, R510, R520, R530, R545, R555, and R560 drivers, which are not forward-compatible with CUDA 12.8. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 25.03 is based on 2.7.0a0+7c8ec84dab.
- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.
- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at /etc/pip/constraint.txt. This file specifies the versions of all python packages used during the PyTorch container creation, and is included to prevent unintentional overwriting of any of the project's dependencies. To install a different version of one of the packages constrained here, the file /etc/pip/constraint.txt within the container must be modified. Simply remove the version constraints for any packages that you want to overwrite, keeping in mind that any other versions than those specified in the constraint file have not been fully tested in the container.

Announcements

- Starting from the 25.03 release, the PyTorch container has implemented a pip constraints file at `/etc/pip/constraint.txt`.
- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.

- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.03	24.04	NVIDIA CUDA 12.8.1.012	2.7.0a0+7c8ec84dab	TensorRT 10.9.0.34
25.02		NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision

training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}
```

```
void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- None.

Chapter 24. PyTorch Release 25.02

The NVIDIA container image for PyTorch, release 25.02 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.12/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 24.04 including Python 3.12
- NVIDIA CUDA 12.8.0.38
- NVIDIA cuBLAS 12.8.3.14
- NVIDIA cuDNN 9.7.1.26
- NVIDIA NCCL 2.25.1
- NVIDIA RAPIDS™ 24.12
- rdma-core 50.0
- NVIDIA HPC-X 2.21
- SHARP 3.9.0
- UCC 1.4.0
- OpenUCX 1.18.0
- OpenMPI 4.1.7
- GDRCopy 2.4.1
- TensorBoard 2.16.2
- Nsight Compute 2025.1.0.14
- Nsight Systems 2025.1.1.65
- NVIDIA TensorRT™ 10.8.0.43
- Torch-TensorRT 2.6.0a0
- NVIDIA DALI® 1.46
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2

- JupyterLab 4.3.5 including Jupyter-TensorBoard
- TransformerEngine 2.0
- TensorRT Model Optimizer 0.23

Driver Requirements

Release 25.02 is based on CUDA 12.8.0.38 which requires NVIDIA Driver release 570 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 470.57 (or later R470), 525.85 (or later R525), 535.86 (or later R535), or 545.23 (or later R545).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R450, R460, R510, R520, R530, R545, R555, and R560 drivers, which are not forward-compatible with CUDA 12.8. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 25.02 is based on 2.7.0a0+6c54963f75.
- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.

Announcements

- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.

- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.02	24.04	NVIDIA CUDA 12.8.0.38	2.7.0a0+6c54963f75	TensorRT 10.8.0.43
25.01		NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()
```

```
# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- cuSPARSELt version featured in 25.02 does not support Blackwell architecture.
- Nvfuser does not currently support sm_120 architecture.

Chapter 25. PyTorch Release 25.01

The NVIDIA container image for PyTorch, release 25.01 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 24.04 including Python 3.12
- NVIDIA CUDA 12.8.0.038
- NVIDIA cuBLAS 12.8.3.14
- NVIDIA cuDNN 9.7.0.66
- NVIDIA NCCL 2.25.1
- NVIDIA RAPIDS™ 24.10
- rdma-core 50.0
- NVIDIA HPC-X 2.21
- SHARP 3.9.0
- UCC 1.4.0
- OpenUCX 1.18.0
- OpenMPI 4.1.7
- GDRCopy 2.4.1
- TensorBoard 2.16.2
- Nsight Compute 2025.1.0.14
- Nsight Systems 2024.6.2.225
- NVIDIA TensorRT™ 10.8.0.40
- Torch-TensorRT 2.6.0a0
- NVIDIA DALI® 1.45
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2

- JupyterLab 4.3.4 including Jupyter-TensorBoard
- TransformerEngine 1.14
- TensorRT Model Optimizer 0.21

Driver Requirements

Release 25.01 is based on CUDA 12.8.0 which requires NVIDIA Driver release 570 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 470.57 (or later R470), 525.85 (or later R525), 535.86 (or later R535), or 545.23 (or later R545).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R450, R460, R510, R520, R530, R545, R555, and R560 drivers, which are not forward-compatible with CUDA 12.8. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 25.01 is based on 2.6.0a0+ecf3bae40a.
- torch.cuda.MemPool() API is no longer experimental and is stable. It enables mixing multiple CUDA system allocators in the same PyTorch program.

Announcements

- Starting with the 25.01 release, the NVIDIA Optimized Deep Learning Framework containers are optimized for Blackwell GPU architectures.
- Volta GPU compute architecture support is discontinued starting with the 25.01 release. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cuSPARSELt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.

- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
25.01	24.04	NVIDIA CUDA 12.8.0	2.6.0a0+ecf3bae40a	TensorRT 10.8.0.40
24.12		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() API

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Get a machine that has NVSwitch, e.g. a DGX H200, and run with upstream nightly
 binaries.

# Note that we need at least 4 GPUs for NVLS reduction.
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 nvls_reduction.py

# Note that NCCL_ALGO=NVLS is to force the usage of NVLS for this demo. You will see
# "NCCL INFO rank 0 successfully local-registered" in the nccl debug output. It
 indicates that NVLS is being
# used. Without NCCL_ALGO set, NCCL will use heuristics to decide whether to run
 with NVLS or not.

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr, size_t size, int device, void* stream) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,

    build_directory="./",    is_python_module=False,
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
).allocator()
```

```
# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator)

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

# register user buffers using ncclCommRegister (called under the hood)
backend.register_mem_pool(pool)

# Collective uses Zero Copy NVLS
dist.all_reduce(tensor[0:4])
torch.cuda.synchronize()
print(tensor[0:4])

# release memory to system
del tensor, del pool
```

Known Issues

- cuSPARSELt version featured in 25.01 does not support Blackwell architecture.

Chapter 26. PyTorch Release 24.12

The NVIDIA container image for PyTorch, release 24.12 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 24.04 including Python 3.12
- NVIDIA CUDA 12.6.3
- NVIDIA cuBLAS 12.6.4.1
- NVIDIA cuDNN 9.6.0.74
- NVIDIA NCCL 2.23.4
- NVIDIA RAPIDS™ 24.10
- rdma-core 39.0
- NVIDIA HPC-X 2.21
- SHARP 3.9.0
- UCC 1.4.0
- OpenUCX 1.18.0
- OpenMPI 4.1.7
- GDRCopy 2.4.1
- TensorBoard 2.16.2
- Nsight Compute 2024.3.2.3
- Nsight Systems 2024.6.1.90
- NVIDIA TensorRT™ 10.7.0.23
- Torch-TensorRT 2.6.0a0
- NVIDIA DALI® 1.44
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2

- JupyterLab 4.2.5 including Jupyter-TensorBoard
- TransformerEngine 1.13
- TensorRT Model Optimizer 0.19.0

Driver Requirements

Release 24.12 is based on CUDA 12.6.3 which requires NVIDIA Driver release 560 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 470.57 (or later R470), 525.85 (or later R525), 535.86 (or later R535), or 545.23 (or later R545).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R450, R460, R510, R520, R530, R545 and R555 drivers, which are not forward-compatible with CUDA 12.6. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 24.12 is based on 2.6.0a0+df5bbc09d1.
- torch.cuda.MemPool() experimental API enables mixing multiple CUDA system allocators in the same PyTorch program.

Announcements

- Starting with the 24.10 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Volta GPU architectures. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise. Volta GPU compute architecture support will be discontinued by the 25.01 release.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cusparse_lt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.

- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.12	24.04	NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.7.0.23
24.11		NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() experimental API

This is an early preview of the API that NVIDIA is working on with the upstream PyTorch community. It is tracked through https://github.com/pytorch/pytorch/issues/124807, and future release notes will include more details.

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions

for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 mempool_example.py

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,
    is_python_module=False,
    build_directory="./",
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
)

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator.allocator())

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
```

```
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

    # register user buffers using ncclCommRegister (called under the hood)
    backend.register_user_buffers(device)

    # Collective uses Zero Copy NVLS
    dist.all_reduce(tensor[0:4])
    torch.cuda.synchronize()
    print(tensor[0:4])

# release memory to system
del tensor
pool.release()
pool.empty_cache()
```

Known Issues

- None.

Chapter 27. PyTorch Release 24.11

The NVIDIA container image for PyTorch, release 24.11 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 24.04 including Python 3.12
- NVIDIA CUDA 12.6.3
- NVIDIA cuBLAS 12.6.4.1
- NVIDIA cuDNN 9.5.1.17
- NVIDIA NCCL 2.22.3
- NVIDIA RAPIDS™ 24.10
- rdma-core 39.0
- NVIDIA HPC-X 2.21
- SHARP 3.9.0
- UCC 1.4.0
- OpenUCX 1.18.0
- OpenMPI 4.1.7
- GDRCopy 2.4.1
- TensorBoard 2.16.2
- Nsight Compute 2024.3.2.3
- Nsight Systems 2024.6.1.90
- NVIDIA TensorRT™ 10.6.0.26
- Torch-TensorRT 2.6.0a0
- NVIDIA DALI® 1.43
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2

- JupyterLab 4.2.5 including Jupyter-TensorBoard
- TransformerEngine 1.12
- TensorRT Model Optimizer 0.19.0

Driver Requirements

Release 24.11 is based on CUDA 12.6.3 which requires NVIDIA Driver release 560 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 470.57 (or later R470), 525.85 (or later R525), 535.86 (or later R535), or 545.23 (or later R545).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R450, R460, R510, R520, R530, R545 and R555 drivers, which are not forward-compatible with CUDA 12.6. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 24.11 is based on 2.6.0a0+df5bbc09d1.
- torch.cuda.MemPool() experimental API enables mixing multiple CUDA system allocators in the same PyTorch program.

Announcements

- Starting with the 24.10 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Volta GPU architectures. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise. Volta GPU compute architecture support will be discontinued by the 25.01 release.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cusparse_lt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.

- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.11	24.04	NVIDIA CUDA 12.6.3	2.6.0a0+df5bbc0	TensorRT 10.6.0.26
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12			1.4.0a0+174e1ba	TensorRT 6.0.1

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
19.11				
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() experimental API

This is an early preview of the API that NVIDIA is working on with the upstream PyTorch community. It is tracked through https://github.com/pytorch/pytorch/issues/124807, and future release notes will include more details.

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-per-node 4 mempool_example.py
```

```
import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,
    is_python_module=False,
    build_directory="./",
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
)

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator.allocator())

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

    # register user buffers using ncclCommRegister (called under the hood)
    backend.register_user_buffers(device)
```

```
    # Collective uses Zero Copy NVLS
    dist.all_reduce(tensor[0:4])
    torch.cuda.synchronize()
    print(tensor[0:4])

# release memory to system
del tensor
pool.release()
pool.empty_cache()
```

Known Issues

- None.

Chapter 28. PyTorch Release 24.10

The NVIDIA container image for PyTorch, release 24.10 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 22.04 including Python 3.10
- NVIDIA CUDA 12.6.2
- NVIDIA cuBLAS 12.6.3.3
- NVIDIA cuDNN 9.5.0.50
- NVIDIA NCCL 2.22.3
- NVIDIA RAPIDS™ 24.08
- rdma-core 39.0
- NVIDIA HPC-X 2.20
- OpenMPI 4.1.7
- GDRCopy 2.3
- TensorBoard 2.16.2
- Nsight Compute 2024.3.2.3
- Nsight Systems 2024.6.1.90
- NVIDIA TensorRT™ 10.5.0.18
- Torch-TensorRT 2.5.0a0
- NVIDIA DALI® 1.42
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2
- JupyterLab 4.2.5 including Jupyter-TensorBoard
- TransformerEngine 1.11
- TensorRT Model Optimizer 0.15.1

Driver Requirements

Release 24.10 is based on CUDA 12.6.2 which requires NVIDIA Driver release 560 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 470.57 (or later R470), 525.85 (or later R525), 535.86 (or later R535), or 545.23 (or later R545).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R450, R460, R510, R520, R530, R545 and R555 drivers, which are not forward-compatible with CUDA 12.6. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 24.10 is based on **2.5.0a0+e000cf0ad9**.
- torch.cuda.MemPool() experimental API enables mixing multiple CUDA system allocators in the same PyTorch program.

Announcements

- Starting with the 24.10 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Volta GPU architectures. For containers that are tested on Volta GPU please refer to NVIDIA AI Enterprise.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cusparse_lt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular

Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.10	22.04	NVIDIA CUDA 12.6.2	2.5.0a0+e000cf0ad9	TensorRT 10.5.0.18
24.09		NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream	TensorRT 5.1.5

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
			commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

torch.cuda.MemPool() experimental API

This is an early preview of the API that NVIDIA is working on with the upstream PyTorch community. It is tracked through https://github.com/pytorch/pytorch/issues/124807, and future release notes will include more details.

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 mempool_example.py

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension
```

```
# create allocator
nccl_allocator_source = """
#include <nccl.h>
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,
    is_python_module=False,
    build_directory="./",
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
)

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator.allocator())

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

    # register user buffers using ncclCommRegister (called under the hood)
    backend.register_user_buffers(device)

    # Collective uses Zero Copy NVLS
    dist.all_reduce(tensor[0:4])
    torch.cuda.synchronize()
    print(tensor[0:4])

# release memory to system
del tensor
pool.release()
pool.empty_cache()
```

Known Issues

- None.

Chapter 29. PyTorch Release 24.09

The NVIDIA container image for PyTorch, release 24.09 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 22.04 including Python 3.10
- NVIDIA CUDA 12.6.1
- NVIDIA cuBLAS 12.6.3.1
- NVIDIA cuDNN 9.4.0.58
- NVIDIA NCCL 2.22.3
- NVIDIA RAPIDS™ 24.08
- rdma-core 39.0
- NVIDIA HPC-X 2.20
- OpenMPI 4.1.7
- GDRCopy 2.3
- TensorBoard 2.16.2
- Nsight Compute 2024.3.1.2
- Nsight Systems 2024.4.2.133
- NVIDIA TensorRT™ 10.4.0.26
- Torch-TensorRT 2.5.0a0
- NVIDIA DALI® 1.41
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2
- JupyterLab 4.2.5 including Jupyter-TensorBoard
- TransformerEngine 1.10
- TensorRT Model Optimizer 0.15.1

Driver Requirements

Release 24.09 is based on CUDA 12.6.1 which requires NVIDIA Driver release 560 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 470.57 (or later R470), 525.85 (or later R525), 535.86 (or later R535), or 545.23 (or later R545).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R450, R460, R510, R520, R530, R545 and R555 drivers, which are not forward-compatible with CUDA 12.6. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 24.09 is based on **2.5.0a0+b465a5843b**.
- torch.cuda.MemPool() experimental API enables mixing multiple CUDA system allocators in the same PyTorch program.

Announcements

- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details. At this point TensorRT Model Optimizer supports x86_64 architecture only and support for other architectures (e.g. ARM64) is experimental.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cusparse_lt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.09	22.04	NVIDIA CUDA 12.6.1	2.5.0a0+b465a5843b	TensorRT 10.4.0.26
24.08		NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as	TensorRT 5.1.5

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
			well as a cherry-picked	

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.

- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.
- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.
- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.
- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.
- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

torch.cuda.MemPool() experimental API

This is an early preview of the API that NVIDIA is working on with the upstream PyTorch community. It is tracked through https://github.com/pytorch/pytorch/issues/124807, and future release notes will include more details.

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 mempool_example.py

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
```

```
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,
    is_python_module=False,
    build_directory="./",
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
)

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator.allocator())

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

    # register user buffers using ncclCommRegister (called under the hood)
    backend.register_user_buffers(device)

    # Collective uses Zero Copy NVLS
    dist.all_reduce(tensor[0:4])
    torch.cuda.synchronize()
    print(tensor[0:4])

# release memory to system
del tensor
pool.release()
pool.empty_cache()
```

Known Issues

- The use of environment variable TORCH_SHOW_CPP_STACKTRACES=1 might cause a stack overflow and a segmentation fault on ARM servers. See stock PyTorch issue for details. This issue has been mitigated via PR134387, pytorch versions later than this commit would no longer encounter this issue.

Chapter 30. PyTorch Release 24.08

The NVIDIA container image for PyTorch, release 24.08 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 22.04 including Python 3.10
- NVIDIA CUDA 12.6
- NVIDIA cuBLAS 12.6.0.22
- NVIDIA cuDNN 9.3.0.75
- NVIDIA NCCL 2.22.3
- NVIDIA RAPIDS™ 24.06
- rdma-core 39.0
- NVIDIA HPC-X 2.19
- OpenMPI 4.1.7
- GDRCopy 2.3
- TensorBoard 2.16.2
- Nsight Compute 2024.3.0.15
- Nsight Systems 2024.4.2.133
- NVIDIA TensorRT™ 10.3.0.26
- Torch-TensorRT 2.5.0a0
- NVIDIA DALI® 1.40
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2
- JupyterLab 4.2.4 including Jupyter-TensorBoard
- TransformerEngine 1.9
- TensorRT Model Optimizer 0.15.0

Driver Requirements

Release 24.08 is based on CUDA 12.6 which requires NVIDIA Driver release 560 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 470.57 (or later R470), 525.85 (or later R525), 535.86 (or later R535), or 545.23 (or later R545).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R450, R460, R510, R520, R530, R545 and R555 drivers, which are not forward-compatible with CUDA 12.6. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 24.08 is based on **2.5.0a0+872d972e41**.
- torch.cuda.MemPool() experimental API enables mixing multiple CUDA system allocators in the same PyTorch program.

Announcements

- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details. At this point TensorRT Model Optimizer supports x86_64 architecture only and support for other architectures (e.g. ARM64) is experimental.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cusparse_lt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.

- A preview of Torch-TensorRT (1.4.0dev0) is now included. Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.08	22.04	NVIDIA CUDA 12.6	2.5.0a0+872d972e41	TensorRT 10.3.0.26
24.07		NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from	TensorRT 5.1.5

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
			July 31, 2019 as well as a cherry-picked	

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.

- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.
- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.
- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.
- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.
- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

torch.cuda.MemPool() experimental API

This is an early preview of the API that NVIDIA is working on with the upstream PyTorch community. It is tracked through https://github.com/pytorch/pytorch/issues/124807, and future release notes will include more details.

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 mempool_example.py

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
```

```
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,
    is_python_module=False,
    build_directory="./",
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
)

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator.allocator())

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

    # register user buffers using ncclCommRegister (called under the hood)
    backend.register_user_buffers(device)

    # Collective uses Zero Copy NVLS
    dist.all_reduce(tensor[0:4])
    torch.cuda.synchronize()
    print(tensor[0:4])

# release memory to system
del tensor
pool.release()
pool.empty_cache()
```

Known Issues

- The use of environment variable TORCH_SHOW_CPP_STACKTRACES=1 might cause a stack overflow and a segmentation fault on ARM servers. See stock PyTorch issue for details.

Chapter 31. PyTorch Release 24.07

The NVIDIA container image for PyTorch, release 24.07 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 22.04 including Python 3.10
- NVIDIA CUDA 12.5.1
- NVIDIA cuBLAS 12.5.3.2
- NVIDIA cuDNN 9.2.1.18
- NVIDIA NCCL 2.22.3
- NVIDIA RAPIDS™ 24.04
- rdma-core 39.0
- NVIDIA HPC-X 2.19
- OpenMPI 4.1.7
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2024.2.1.2
- Nsight Systems 2024.4.2.133
- NVIDIA TensorRT™ 10.2.0.19
- Torch-TensorRT 2.4.0a0
- NVIDIA DALI® 1.39
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 1.8
- PyTorch quantization wheel 2.1.2

Driver Requirements

Release 24.07 is based on CUDA 12.5.1 which requires NVIDIA Driver release 555 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 470.57 (or later R470), 525.85 (or later R525), 535.86 (or later R535), or 545.23 (or later R545).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R450, R460, R510, R520 and R545 drivers, which are not forward-compatible with CUDA 12.5. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 24.07 is based on **2.4.0a0+3bcc3cddb5**.
- torch.cuda.MemPool() experimental API enables mixing multiple CUDA system allocators in the same PyTorch program.

Announcements

- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use `pip list |grep modelopt` to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cusparse_lt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.

- A preview of Torch-TensorRT (1.4.0dev0) is now included. Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.07	22.04	NVIDIA CUDA 12.5.1	2.4.0a0+3bcc3cddb5	TensorRT 10.2.0.19
24.06		NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05		NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as	TensorRT 5.1.5

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
			well as a cherry-picked	

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.

- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.
- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.
- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.
- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.
- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

torch.cuda.MemPool() experimental API

This is an early preview of the API that NVIDIA is working on with the upstream PyTorch community. It is tracked through https://github.com/pytorch/pytorch/issues/124807, and future release notes will include more details.

torch.cuda.MemPool() enables usage of multiple CUDA system allocators in the same PyTorch program. Following is an example that enables NVLink Sharp (NVLS) reductions for part of a PyTorch program, by using ncclMemAlloc allocator, and user buffer registration using ncclCommRegister.

```
# Run with NCCL_ALGO=NVLS NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=NVLS torchrun --nproc-
per-node 4 mempool_example.py

import os

import torch
import torch.distributed as dist
from torch.cuda.memory import CUDAPluggableAllocator
from torch.distributed.distributed_c10d import _get_default_group
from torch.utils import cpp_extension


# create allocator
nccl_allocator_source = """
#include <nccl.h>
```

```
#include <iostream>
extern "C" {

void* nccl_alloc_plug(size_t size, int device, void* stream) {
  std::cout << "Using ncclMemAlloc" << std::endl;
  void* ptr;
  ncclResult_t err = ncclMemAlloc(&ptr, size);
  return ptr;

}

void nccl_free_plug(void* ptr) {
  std::cout << "Using ncclMemFree" << std::endl;
  ncclResult_t err = ncclMemFree(ptr);
}

}
"""
nccl_allocator_libname = "nccl_allocator"
nccl_allocator = torch.utils.cpp_extension.load_inline(
    name=nccl_allocator_libname,
    cpp_sources=nccl_allocator_source,
    with_cuda=True,
    extra_ldflags=["-lnccl"],
    verbose=True,
    is_python_module=False,
    build_directory="./",
)

allocator = CUDAPluggableAllocator(
    f"./{nccl_allocator_libname}.so", "nccl_alloc_plug", "nccl_free_plug"
)

# setup distributed
rank = int(os.getenv("RANK"))
local_rank = int(os.getenv("LOCAL_RANK"))
world_size = int(os.getenv("WORLD_SIZE"))
torch.cuda.set_device(local_rank)
dist.init_process_group(backend="nccl")
device = torch.device(f"cuda:{local_rank}")
default_pg = _get_default_group()
backend = default_pg._get_backend(device)

# create pool
pool = torch.cuda.MemPool(allocator.allocator())

with torch.cuda.use_mem_pool(pool):
    # tensor gets allocated with ncclMemAlloc passed in the pool
    tensor = torch.arange(1024 * 1024 * 2, device=device)
    print(f"tensor ptr on rank {rank} is {hex(tensor.data_ptr())}")

    # register user buffers using ncclCommRegister (called under the hood)
    backend.register_user_buffers(device)

    # Collective uses Zero Copy NVLS
    dist.all_reduce(tensor[0:4])
    torch.cuda.synchronize()
    print(tensor[0:4])

# release memory to system
del tensor
pool.release()
pool.empty_cache()
```

Known Issues

- Inference performance of BERT, Conformer, and ViT models regressed, fix is WIP.

Chapter 32. PyTorch Release 24.06

The NVIDIA container image for PyTorch, release 24.06 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 22.04 including Python 3.10
- NVIDIA CUDA 12.5.0.23
- NVIDIA cuBLAS 12.5.2.13
- NVIDIA cuDNN 9.1.0.70
- NVIDIA NCCL 2.21.5
- NVIDIA RAPIDS™ 24.04
- rdma-core 39.0
- NVIDIA HPC-X 2.19
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2024.2.0.16
- Nsight Systems 2024.2.3.38
- NVIDIA TensorRT™ 10.1.0.27
- Torch-TensorRT 2.4.0a0
- NVIDIA DALI® 1.38
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 1.7
- PyTorch quantization wheel 2.1.2

Driver Requirements

Release 24.06 is based on NVIDIA CUDA 12.5.0.23, which requires NVIDIA Driver release 545 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 470.57 (or later R470), 525.85 (or later R525), 535.86 (or later R535), or 545.23 (or later R545).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R450, R460, R510, and R520 drivers, which are not forward-compatible with CUDA 12.3. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 24.06 is based on **2.4.0a0+f70bd71a48**.

Announcements

- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release ships with TensorRT Model Optimizer, use "*pip list nvidia-modelopt*" to check version details.
- Starting with the 24.06 release, the NVIDIA Optimized PyTorch container release builds pytorch with cusparse_lt turned-on, similar to stock PyTorch.
- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- A preview of Torch-TensorRT (1.4.0dev0) is now included. Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed

and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.

- Starting with the 24.05 release, `torchtext` and `torchdata` have been removed in the NGC PyTorch container.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.06	22.04	NVIDIA CUDA 12.5.0.23	2.4.0a0+f70bd71a48	TensorRT 10.1.0.27
24.05	22.04	NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the

overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

This model script is available on GitHub and NGC.

- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.

- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao,

Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.

- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- Inference performance of BERT, Conformer, and ViT models regressed, fix is WIP.

Chapter 33. PyTorch Release 24.05

The NVIDIA container image for PyTorch release 24.05 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 22.04 including Python 3.10
- NVIDIA CUDA 12.4.1
- NVIDIA cuBLAS 12.4.5.8
- NVIDIA cuDNN 9.1.0.70
- NVIDIA NCCL 2.21.5
- NVIDIA RAPIDS™ 24.04
- rdma-core 39.0
- NVIDIA HPC-X 2.19
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2024.1.14
- Nsight Systems 2024.2.1.106
- NVIDIA TensorRT™ 10.0.1.6
- Torch-TensorRT 2.4.0a0
- NVIDIA DALI® 1.37
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 1.6
- PyTorch quantization wheel 2.1.2

Driver Requirements

Release 24.05 is based on NVIDIA CUDA 12.4.1, which requires NVIDIA Driver release 545 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 470.57 (or later R470), 525.85 (or later R525), 535.86 (or later R535), or 545.23 (or later R545).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R450, R460, R510, and R520 drivers, which are not forward-compatible with CUDA 12.3. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 24.05 is based on **2.4.0a0+07cecf4168**.

Announcements

- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder (/opt/pytorch/lightning-thunder).
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- A preview of Torch-TensorRT (1.4.0dev0) is now included. Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example,

using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.

- The deprecation process for `torchtext` and `torchdata` started with the 24.02 PyTorch NGC container and is now complete with this release.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.05	22.04	NVIDIA CUDA 12.4.1	2.4.0a0+07cecf4168	TensorRT 10.0.1.6
24.04			2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.

- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.
- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.
- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.
- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- Inference performance of BERT, Conformer, and ViT models regressed, fix is WIP.

Chapter 34. PyTorch Release 24.04

The NVIDIA container image for PyTorch, release 24.04 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 22.04 including Python 3.10
- NVIDIA CUDA 12.4
- NVIDIA cuBLAS 12.4.5.8
- NVIDIA cuDNN 9.1.0.70
- NVIDIA NCCL 2.21.5
- NVIDIA RAPIDS™ 24.02
- rdma-core 39.0
- NVIDIA HPC-X 2.18
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2024.1.0.13
- Nsight Systems 2024.2.1.38
- NVIDIA TensorRT™ 8.6.3
- Torch-TensorRT 2.3.0a0
- NVIDIA DALI® 1.36
- nvImageCodec 0.2.0.7
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 1.5
- PyTorch quantization wheel 2.1.2

Driver Requirements

Release 24.04 is based on NVIDIA CUDA 12.4.1, which requires NVIDIA Driver release 545 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 470.57 (or later R470), 525.85 (or later R525), 535.86 (or later R535), or 545.23 (or later R545).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R450, R460, R510, and R520 drivers, which are not forward-compatible with CUDA 12.3. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 24.04 is based on 2.3.0a0+6ddf5cf85e.

Announcements

- Starting with the 24.03 release, the NVIDIA Optimized PyTorch container release provides access to lightning-thunder.
- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- A preview of Torch-TensorRT (1.4.0dev0) is now included. Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0. Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example,

using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`. Starting with the 24.02 PyTorch NGC container, we have started the deprecation process on torchtext and torchdata. Both packages will be removed starting with release 24.05.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.04	22.04	NVIDIA CUDA 12.4.1	2.3.0a0+6ddf5cf85e	TensorRT 8.6.3
24.03		NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.

- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.
- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.
- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.
- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- Up to 95% performance regression for TFT inference on Volta GPUs if TorchScript with nvFuser is enabled. Disable TorchScript or use an alternative backend to reduce the regression.
- Up to 40% performance regression for BERT models on H100 GPUs may be observed if TorchScript with nvFuser is enabled. Disable TorchScript or use an alternative backend to reduce the regression.
- UNet models running on A100 GPUs in distributed settings may have up to ~3x slowdown.

Chapter 35. PyTorch Release 24.03

The NVIDIA container image for PyTorch, release 24.03 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 22.04 including Python 3.10
- NVIDIA CUDA 12.4.0.41
- NVIDIA cuBLAS 12.4.2.65
- NVIDIA cuDNN 9.0.0.306
- NVIDIA NCCL 2.20
- NVIDIA RAPIDS™ 24.02
- rdma-core 39.0
- NVIDIA HPC-X 2.18
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2024.1.0.13
- Nsight Systems 2024.2.1.38
- NVIDIA TensorRT™ 8.6.3
- Torch-TensorRT 2.3.0a0
- NVIDIA DALI® 1.35
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 1.4
- PyTorch quantization wheel 2.1.2

Driver Requirements

Release 24.03 is based on CUDA 12.4.0.41, which requires NVIDIA Driver release 545 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 470.57 (or later R470), 525.85 (or later R525), 535.86 (or later R535), or 545.23 (or later R545).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R450, R460, R510, and R520 drivers, which are not forward-compatible with CUDA 12.3. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 24.03 is based on 2.3.0a0+40ec155e58.

Announcements

- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.11 release, NVIDIA PyTorch containers supporting integrated GPU embedded systems will be published.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- A preview of Torch-TensorRT (1.4.0dev0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.

- Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.03 PyTorch NGC container, we have started the deprecation process on torchtext and torchdata. Both packages will be removed starting with release 24.05.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.03	22.04	NVIDIA CUDA 12.4.0.41	2.3.0a0+40ec155e58	TensorRT 8.6.3
24.02		NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with

FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

This model script is available on GitHub and NGC.

- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.

- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao,

Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.

- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- There is a known CUPTI permissions issue that is new in the 24.01 iGPU container. This issue prevents the profiler from being able to capture cuda events and manifests itself as a CUPTI Runtime Error with error code 35. This may be worked around by running the following command:

  ```
  rm -rf /usr/local/cuda/compat/lib.real
  ```

- Up to 95% performance regression for TFT inference on Volta GPUs if TorchScript with nvFuser is enabled. Disable TorchScript or use an alternative backend to reduce the regression.
- Up to 40% performance regression for BERT models on H100 GPUs may be observed if TorchScript with nvFuser is enabled. Disable TorchScript or use an alternative backend to reduce the regression.

Chapter 36. PyTorch Release 24.02

The NVIDIA container image for PyTorch, release 24.02 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 22.04 including Python 3.10
- NVIDIA CUDA 12.3.2
- NVIDIA cuBLAS 12.3.4.1
- NVIDIA cuDNN 9.0.0.306
- NVIDIA NCCL 2.19.4
- NVIDIA RAPIDS™ 23.12
- rdma-core 39.0
- NVIDIA HPC-X 2.16rc4
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2023.3.1.1
- Nsight Systems 2023.4.1.97
- NVIDIA TensorRT™ 8.6.3
- Torch-TensorRT 2.2.0a0
- NVIDIA DALI® 1.34
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 1.3
- PyTorch quantization wheel 2.1.2

Driver Requirements

Release 24.02 is based on CUDA 12.3.2, which requires NVIDIA Driver release 545 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 470.57 (or later R470), 525.85 (or later R525), 535.86 (or later R535), or 545.23 (or later R545).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R450, R460, R510, and R520 drivers, which are not forward-compatible with CUDA 12.3. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 24.02 is based on 2.3.0a0+ebedce2.

Announcements

- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.11 release, NVIDIA PyTorch containers supporting integrated GPU embedded systems will be published.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- A preview of Torch-TensorRT (1.4.0dev0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.

- Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.
- Starting with the 24.02 PyTorch NGC container, we have started the deprecation process on torchtext and torchdata. Both packages will be removed starting with release 24.05.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.02	22.04	NVIDIA CUDA 12.3.2	2.3.0a0+ebedce2	TensorRT 8.6.3
24.01			2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.

- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.
- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.
- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.
- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- There is a known CUPTI permissions issue that is new in the 24.01 iGPU container. This issue prevents the profiler from being able to capture cuda events and manifests itself as a CUPTI Runtime Error with error code 35. This may be worked around by running the following command:

  ```
  rm -rf /usr/local/cuda/compat/lib.real
  ```
- Up to 95% performance regression for TFT inference on Volta GPUs if TorchScript with nvFuser is enabled. Disable TorchScript or use an alternative backend to reduce the regression.
- Up to 40% performance regression for BERT models on H100 GPUs may be observed if TorchScript with nvFuser is enabled. Disable TorchScript or use an alternative backend to reduce the regression.

Chapter 37. PyTorch Release 24.01

The NVIDIA container image for PyTorch, release 24.01 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 22.04 including Python 3.10
- NVIDIA CUDA 12.3.2
- NVIDIA cuBLAS 12.3.4.1
- NVIDIA cuDNN 8.9.7.29
- NVIDIA NCCL 2.19.4
- NVIDIA RAPIDS™ 23.12
- rdma-core 39.0
- NVIDIA HPC-X 2.16rc4
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2023.3.1.1
- Nsight Systems 2023.4.1.97
- NVIDIA TensorRT™ 8.6.1.6
- Torch-TensorRT 2.2.0a0
- NVIDIA DALI® 1.33
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine v1.2.1
- PyTorch quantization wheel 2.1.2

Driver Requirements

Release 24.01 is based on CUDA 12.3.2, which requires NVIDIA Driver release 545 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 470.57 (or later R470), 525.85 (or later R525), 535.86 (or later R535), or 545.23 (or later R545).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R450, R460, R510, and R520 drivers, which are not forward-compatible with CUDA 12.3. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 24.01 is based on 2.2.0a0+81ea7a4.

Announcements

- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.11 release, NVIDIA PyTorch containers supporting integrated GPU embedded systems will be published.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- A preview of Torch-TensorRT (1.4.0dev0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.

- Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
24.01	22.04	NVIDIA CUDA 12.3.2	2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.12				
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.

- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.
- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.

- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.

- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- There is a known CUPTI permissions issue that is new in the 24.01 iGPU container. This issue prevents the profiler from being able to capture cuda events and manifests itself as a CUPTI Runtime Error with error code 35. This may be worked around by running the following command:

  ```
  rm -rf /usr/local/cuda/compat/lib.real
  ```

- Up to 95% performance regression for TFT inference on Volta GPUs if TorchScript with nvFuser is enabled. Disable TorchScript or use an alternative backend to reduce the regression.
- Up to 40% performance regression for BERT models on H100 GPUs may be observed if TorchScript with nvFuser is enabled. Disable TorchScript or use an alternative backend to reduce the regression.

Chapter 38. PyTorch Release 23.12

The NVIDIA container image for PyTorch, release 23.12 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 22.04 including Python 3.10
- NVIDIA CUDA 12.3.2
- NVIDIA cuBLAS 12.3.4.1
- NVIDIA cuDNN 8.9.7.29
- NVIDIA NCCL 2.19.3
- NVIDIA RAPIDS™ 23.10
- rdma-core 39.0
- NVIDIA HPC-X 2.16
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2023.3.1.1
- Nsight Systems 2023.3.4.1
- NVIDIA TensorRT™ 8.6.1.6
- Torch-TensorRT 2.2.0a0
- NVIDIA DALI® 1.32
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 1.1
- PyTorch quantization wheel 2.1.2

Driver Requirements

Release 23.12 is based on CUDA 12.3.2, which requires NVIDIA Driver release 545 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), 510.47 (or later R510), or 525.85 (or later R525) 535.86 (or later R535), or 545.23 (or later R545).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R460, and R520 drivers, which are not forward-compatible with CUDA 12.3. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 23.12 is based on commit 81ea7a489a85d6f6de2c3b63206ca090927e203a s.

Announcements

- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.11 release, NVIDIA PyTorch containers supporting integrated GPU embedded systems will be published.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- A preview of Torch-TensorRT (1.4.0dev0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.

- Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.12	22.04	NVIDIA CUDA 12.3.2	2.2.0a0+81ea7a4	TensorRT 8.6.1.6
23.11		NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.

- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.
- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.

- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.

- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- Up to 95% performance regression for TFT inference on Volta GPUs if TorchScript with nvFuser is enabled. Disable TorchScript or use an alternative backend to reduce the regression.
- Up to 40% performance regression for BERT models on H100 GPUs may be observed if TorchScript with nvFuser is enabled. Disable TorchScript or use an alternative backend to reduce the regression.

Chapter 39. PyTorch Release 23.11

The NVIDIA container image for PyTorch, release 23.11 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 22.04 including Python 3.10
- NVIDIA CUDA 12.3.0
- NVIDIA cuBLAS 12.3.2.9
- NVIDIA cuDNN 8.9.6
- NVIDIA NCCL 2.19.3
- NVIDIA RAPIDS™ 23.08
- Apex
- rdma-core 39.0
- NVIDIA HPC-X 2.15+
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2023.3.0.12
- Nsight Systems 2023.3.1.92
- NVIDIA TensorRT™ 8.6.1.6
- Torch-TensorRT 2.0.0.dev0
- NVIDIA DALI® 1.31.0
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 1.0.0+66d91d5
- PyTorch quantization wheel 2.1.2

Driver Requirements

Release 23.11 is based on CUDA 12.3.0, which requires NVIDIA Driver release 545 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), 510.47 (or later R510), or 525.85 (or later R525) 535.86 (or later R535), or 545.23 (or later R545).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R460, and R520 drivers, which are not forward-compatible with CUDA 12.3. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 23.11 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, NVIDIA Ampere architecture, and NVIDIA Hopper™ architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 23.11 is based on 2.2.0a0+6a974be torch.sparse now includes prototype support for semi-structured (2:4) sparsity on NVIDIA® GPUs.
- nvFuser for TorchScript is now deprecated
- Numpy is upgraded to 1.24.4 in lieu of deprecation

Announcements

- Starting with the 23.11 release, NVIDIA Optimized PyTorch containers supporting iGPU architectures are published, and able to run on Jetson devices. Please refer to the Frameworks Support Matrix for information regarding which iGPU hardware/software is supported by which container.
- Starting with the 23.11 release, NVIDIA PyTorch containers supporting integrated GPU embedded systems will be published.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular

Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.

- A preview of Torch-TensorRT (1.4.0dev0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.11	22.04	NVIDIA CUDA 12.3.0	2.2.0a0+6a974bec	TensorRT 8.6.1.6
23.10		NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation (torch.cuda.amp). AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the

overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

This model script is available on GitHub and NGC.

- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.

- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao,

Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.

- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- Up to 95% performance regression for TFT inference on Volta GPUs if TorchScript with nvFuser is enabled. Disable TorchScript or use an alternative backend to reduce the regression.
- Up to 40% performance regression for BERT models on H100 GPUs may be observed if TorchScript with nvFuser is enabled. Disable TorchScript or use an alternative backend to reduce the regression.

Chapter 40. PyTorch Release 23.10

The NVIDIA container image for PyTorch, release 23.10 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 22.04 including Python 3.10
- NVIDIA CUDA® 12.2.2
- NVIDIA cuBLAS 12.2.5.6
- NVIDIA cuDNN 8.9.5
- NVIDIA NCCL 2.19.3
- NVIDIA RAPIDS™ 23.08
- Apex
- rdma-core 39.0
- NVIDIA HPC-X 2.15+
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2023.2.1.3
- Nsight Systems 2023.3.1.92
- NVIDIA TensorRT™ 8.6.1.6
- Torch-TensorRT 2.0.0.dev0
- NVIDIA DALI® 1.30.0
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 0.12+170797
- PyTorch quantization wheel 2.1.3

Driver Requirements

Release 23.10 is based on CUDA 12.2.2, which requires NVIDIA Driver release 535 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), 510.47 (or later R510), or 525.85 (or later R525), or 535.86 (or later R535).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R460, and R520 drivers, which are not forward-compatible with CUDA 12.2. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 23.10 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, NVIDIA Ampere architecture, and NVIDIA Hopper™ architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 23.10 is based on 2.1.0a0+32f93b1.
- Performance optimizations for Grace Hopper systems.
- nvFUSER is now available in /opt/pytorch
- Fixed iGPU OpenBLAS issues

Announcements

- Starting with the 23.10 release, NVIDIA PyTorch containers supporting integrated GPU embedded systems will be published.
- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- A preview of Torch-TensorRT (1.4.0dev0) is now included.

Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.

- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.10/site-packages"` if your Conda package manager was installed in `/opt/conda`.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.10	22.04	NVIDIA CUDA 12.2.2	2.1.0a0+32f93b1	TensorRT 8.6.1.6
23.09		NVIDIA CUDA 12.2.1		
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation. AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.

- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.
- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.

- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.

- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- Up to 95% performance regression for TFT inference on Volta GPUs if TorchScript with nvFuser is enabled. Disable TorchScript or use an alternative backend to reduce the regression.

Chapter 41. PyTorch Release 23.09

The NVIDIA container image for PyTorch, release 23.09 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 22.04 including Python 3.10
- NVIDIA CUDA® 12.2.1
- NVIDIA cuBLAS 12.2.5.6
- NVIDIA cuDNN 8.9.5
- NVIDIA NCCL 2.18.5
- NVIDIA RAPIDS™ 23.08
- Apex
- rdma-core 39.0
- NVIDIA HPC-X 2.16
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2023.2.1.3
- Nsight Systems 2023.3.1.92
- NVIDIA TensorRT™ 8.6.1.6
- Torch-TensorRT 2.0.0.dev0
- NVIDIA DALI® 1.29.0
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 0.12
- PyTorch quantization wheel 2.1.2

Driver Requirements

Release 23.09 is based on CUDA 12.2.1, which requires NVIDIA Driver release 535 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), 510.47 (or later R510), or 525.85 (or later R525), or 535.86 (or later R535).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R460, and R520 drivers, which are not forward-compatible with CUDA 12.2. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 23.09 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, NVIDIA Ampere architecture, and NVIDIA Hopper™ architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 23.09 is based on 2.1.0a0+32f93b1.

Announcements

- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- A preview of Torch-TensorRT (1.4.0dev0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.

- Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.8/site-packages"` if your Conda package manager was installed in `/opt/conda`.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.09	22.04	NVIDIA CUDA 12.2.1	2.1.0a0+32f93b1	TensorRT 8.6.1.6
23.08			2.1.0a0+29c30b1	
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation. AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.
- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized

version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.

- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.

- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

This model script is available on GitHub and NGC.

Known Issues

- The following CVEs might be flagged but were patched by backporting the fixes into the corresponding libraries in our release:
  - CVE-2022-45198 - Pillow before 9.2.0 performs Improper Handling of Highly Compressed GIF Data (Data Amplification).
  - CVE-2022-45199 - Pillow before 9.3.0 allows denial of service via SAMPLESPERPIXEL.
- Up to 20% performance regression for inference use cases on Volta GPUs.

Chapter 42. PyTorch Release 23.08

The NVIDIA container image for PyTorch, release 23.08 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 22.04 including Python 3.10
- NVIDIA CUDA® 12.2.1
- NVIDIA cuBLAS 12.2.5.1
- NVIDIA cuDNN 8.9.4
- NVIDIA NCCL 2.18.3
- NVIDIA RAPIDS™ 23.06
- Apex
- rdma-core 39.0
- NVIDIA HPC-X 2.15
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2023.2.1.3
- Nsight Systems 2023.2.3.1001
- NVIDIA TensorRT™ 8.6.1.6
- Torch-TensorRT 2.0.0.dev0
- NVIDIA DALI® 1.28.0
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 0.11.0++3f01b4f
- PyTorch quantization wheel 2.1.2

Driver Requirements

Release 23.08 is based on CUDA 12.2.1, which requires NVIDIA Driver release 535 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), 510.47 (or later R510), or 525.85 (or later R525), or 535.86 (or later R535).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R460, and R520 drivers, which are not forward-compatible with CUDA 12.2. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 23.08 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, NVIDIA Ampere architecture, and NVIDIA Hopper™ architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 23.08 is based on 2.1.0a0+29c30b1.

Announcements

- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- A preview of Torch-TensorRT (1.4.0dev0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.

- Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.8/site-packages"` if your Conda package manager was installed in `/opt/conda`.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.08	22.04	NVIDIA CUDA 12.2.1	2.1.0a0+29c30b1	TensorRT 8.6.1.6
23.07		NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation. AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.
- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized

version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.

- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.

- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

This model script is available on GitHub and NGC.

Known Issues

- The following CVEs might be flagged but were patched by backporting the fixes into the corresponding libraries in our release:
  - CVE-2022-45198 - Pillow before 9.2.0 performs Improper Handling of Highly Compressed GIF Data (Data Amplification).
  - CVE-2022-45199 - Pillow before 9.3.0 allows denial of service via SAMPLESPERPIXEL.
- Up to 20% performance regression for inference use cases on Volta GPUs.

Chapter 43. PyTorch Release 23.07

The NVIDIA container image for PyTorch, release 23.07 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 22.04 including Python 3.10
- NVIDIA CUDA® 12.1.1
- NVIDIA cuBLAS 12.1.3.1
- NVIDIA cuDNN 8.9.3
- NVIDIA NCCL 2.18.3
- NVIDIA RAPIDS™ 23.06
- Apex
- rdma-core 39.0
- NVIDIA HPC-X 2.15
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2023.1.1.4
- Nsight Systems 2023.2.3.1001
- NVIDIA TensorRT™ 8.6.1.6
- Torch-TensorRT 1.5.0.dev0
- NVIDIA DALI® 1.27.0
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 0.10.0+96ed6fc
- PyTorch quantization wheel 2.1.2

Driver Requirements

Release 23.07 is based on CUDA 12.1.1, which requires NVIDIA Driver release 530 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), 510.47 (or later R510), 515.65 (or later R515), 525.85 (or later R525), or 530.30 (or later R530).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R460, and R520 drivers, which are not forward-compatible with CUDA 12.1. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 23.07 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, NVIDIA Ampere architecture, and NVIDIA Hopper™ architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 23.07 is based on 2.1.0a0+b5021ba.

Announcements

- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- A preview of Torch-TensorRT (1.4.0dev0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.

- Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.8/site-packages"` if your Conda package manager was installed in `/opt/conda`.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.07	22.04	NVIDIA CUDA 12.1.1	2.1.0a0+b5021ba	TensorRT 8.6.1.6
23.06			2.1.0a0+4136153	
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation. AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores.

This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.
- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:
  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.
- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.
- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.
- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.
- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- The following CVEs might be flagged but were patched by backporting the fixes into the corresponding libraries in our release:

- CVE-2022-45198 - Pillow before 9.2.0 performs Improper Handling of Highly Compressed GIF Data (Data Amplification).
- CVE-2022-45199 - Pillow before 9.3.0 allows denial of service via SAMPLESPERPIXEL.

Chapter 44. PyTorch Release 23.06

The NVIDIA container image for PyTorch, release 23.06 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 22.04 including Python 3.10
- NVIDIA CUDA® 12.1.1
- NVIDIA cuBLAS 12.1.3.1
- NVIDIA cuDNN 8.9.2
- NVIDIA NCCL 2.18.1
- NVIDIA RAPIDS™ 23.04
- Apex
- rdma-core 39.0
- NVIDIA HPC-X 2.15
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2023.1.1.4
- Nsight Systems 2023.2.3.1001
- NVIDIA TensorRT™ 8.6.1.6
- Torch-TensorRT 1.5.0.dev0
- NVIDIA DALI® 1.26.0
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 0.9.0
- PyTorch quantization wheel 2.1.2

Driver Requirements

Release 23.06 is based on CUDA 12.1.1, which requires NVIDIA Driver release 530 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), 510.47 (or later R510), 515.65 (or later R515), 525.85 (or later R525), or 530.30 (or later R530).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R460, and R520 drivers, which are not forward-compatible with CUDA 12.1. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 23.06 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, NVIDIA Ampere architecture, and NVIDIA Hopper™ architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 23.06 is based on 2.1.0a0+4136153.
- 23.06 provides the experimental UCC process group for the distributed backend. Users can experiment with it by creating UCC as the default process group via: `torch.distributed.init_process_group(backend="ucc", kwargs)` or a side process group with any default via:

```
torch.distributed.init_process_group(backend=any_backend,
default_pg_kwargs)

ucc_pg = torch.distributed.new_group(backend="ucc", ucc_pg_kwargs)
```

Announcements

- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers are no longer tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.

- A preview of Torch-TensorRT (1.4.0dev0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.8/site-packages"` if your Conda package manager was installed in `/opt/conda`.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.06	22.04	NVIDIA CUDA 12.1.1	2.1.0a0+4136153	TensorRT 8.6.1.6
23.05			2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation and a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.
- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized

version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.

- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.

- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

This model script is available on GitHub and NGC.

Known Issues

- The following CVEs might be flaggted but were patched by backporting the fixes into the corresponding libraries in our release:
  - CVE-2022-45198 - Pillow before 9.2.0 performs Improper Handling of Highly Compressed GIF Data (Data Amplification).
  - CVE-2022-45199 - Pillow before 9.3.0 allows denial of service via SAMPLESPERPIXEL.
- Performance regression up to 28% for FastPitch on NVIDIA Ampere architecture and Hopper GPUs.
- A full iteration CUDA graph capture including gradient AllReduce, Optimizer, and Parameter AllGather operations could fail with a CUDA error. We recommend reducing the scope of the CUDA graph capture as a workaround.

Chapter 45. PyTorch Release 23.05

The NVIDIA container image for PyTorch, release 23.05 is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.10/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 22.04 including Python 3.10
- NVIDIA CUDA® 12.1.1
- NVIDIA cuBLAS 12.1.3.1
- NVIDIA cuDNN 8.9.1.23
- NVIDIA NCCL 2.18.1
- NVIDIA RAPIDS™ 23.04
- Apex
- rdma-core 36.0
- NVIDIA HPC-X 2.14
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2023.1.1.4
- Nsight Systems 2023.2.1.89
- NVIDIA TensorRT™ 8.6.1.2
- Torch-TensorRT 1.4.0.dev0
- NVIDIA DALI® 1.25.0
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 0.8
- PyTorch quantization wheel 2.1.2

Driver Requirements

Release 23.05 is based on CUDA 12.1.1, which requires NVIDIA Driver release 530 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), 510.47 (or later R510), 515.65 (or later R515), 525.85 (or later R525), or 530.30 (or later R530).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R460, and R520 drivers, which are not forward-compatible with CUDA 12.1. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 23.05 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, NVIDIA Ampere architecture, and NVIDIA Hopper™ architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 23.05 is based on 2.0.0with a cherry-pick to fix potential performance regressions via #97838.
- The 23.05 container updates to Ubuntu 22.04 and Python to 3.10.6.

Announcements

- Starting with the 23.06 release, the NVIDIA Optimized Deep Learning Framework containers will no longer be tested on Pascal GPU architectures.
- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- NVIDIA Deep Learning Profiler (DLProf) v1.8, which was included in the 21.12 container, was the final release of DLProf.

  Starting with the 22.01 container, DLProf is no longer included, but it can still be manually installed by using a pip wheel on the `nvidia-pyindex`.
- A preview of Torch-TensorRT (1.4.0dev0) is now included.

Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.

- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container.

  To profile models in PyTorch, use DLProf.

  DLProf can help data scientists, engineers, and researchers understand improve performance of their models with visualization by using the DLProf Viewer in a web browser or by analyzing text reports. DL Prof is available on NGC or through a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`).

  You can obtain the models from Github or the NVIDIA GPU Cloud (NGC) instead. Some Python packages that were included in previous containers to support these example models have also been removed. Depending on their specific use cases, you might need to add some packages that were previously preinstalled.
- Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.8/site-packages"` if your Conda package manager was installed in `/opt/conda`.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.05	22.04	NVIDIA CUDA 12.1.1	2.0.0	TensorRT 8.6.1.2
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation and a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing

the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

This model script is available on GitHub and NGC.

- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.

- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao,

Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.

- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- The following CVEs might be flaggted but were patched by backporting the fixes into the corresponding libraries in our release:
  - CVE-2022-45198 - Pillow before 9.2.0 performs Improper Handling of Highly Compressed GIF Data (Data Amplification).
  - CVE-2022-45199 - Pillow before 9.3.0 allows denial of service via SAMPLESPERPIXEL.
- The following CVEs were detected in OpenCV: CVE-2023-2618, CVE-2023-2617, which will be addressed in future releases.
- Performance regression up to 35% for FastPitch on Volta GPUs.

Chapter 46. PyTorch Release 23.04

The NVIDIA container image for PyTorch, release 23.04, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.8/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8
- NVIDIA CUDA® 12.1.0
- NVIDIA cuBLAS 12.1.3
- NVIDIA cuDNN 8.9.0
- NVIDIA NCCL 2.17.1
- NVIDIA RAPIDS™ 23.02
- Apex
- rdma-core 36.0
- NVIDIA HPC-X 2.13
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2023.1.0.15
- Nsight Systems 2023.1.1.127
- NVIDIA TensorRT™ 8.6.1
- Torch-TensorRT 1.4.0.dev0
- NVIDIA DALI® 1.23.0
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 0.7
- PyTorch quantization wheel 2.1.2

Driver Requirements

Release 23.04 is based on CUDA 12.1.0, which requires NVIDIA Driver release 530 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), 510.47 (or later R510), 515.65 (or later R515), 525.85 (or later R525), or 530.30 (or later R530).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R460, and R520 drivers, which are not forward-compatible with CUDA 12.1. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 23.04 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, NVIDIA Ampere architecture, and NVIDIA Hopper™ architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 23.04 is based on 2.1.0a0+fe05266f.
- Configuration of NCCL communicators via: https://github.com/pytorch/pytorch/pull/97394

Announcements

- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- NVIDIA Deep Learning Profiler (DLProf) v1.8, which was included in the 21.12 container, was the final release of DLProf.

  Starting with the 22.01 container, DLProf is no longer included, but it can still be manually installed by using a pip wheel on the `nvidia-pyindex`.
- A preview of Torch-TensorRT (1.4.0dev0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.

- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container.

  To profile models in PyTorch, use DLProf.

  DLProf can help data scientists, engineers, and researchers understand improve performance of their models with visualization by using the DLProf Viewer in a web browser or by analyzing text reports. DL Prof is available on NGC or through a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`).

  You can obtain the models from Github or the NVIDIA GPU Cloud (NGC) instead. Some Python packages that were included in previous containers to support these example models have also been removed. Depending on their specific use cases, you might need to add some packages that were previously preinstalled.
- Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.8/site-packages"` if your Conda package manager was installed in `/opt/conda`.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.04	20.04	NVIDIA CUDA 12.1.0	2.1.0a0+fe05266f	TensorRT 8.6.1
23.03			2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation and a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.

- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.
- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.
- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.
- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- The following CVEs might be flaggted but were patched by backporting the fixes into the corresponding libraries in our release:
  - CVE-2022-45198 - Pillow before 9.2.0 performs Improper Handling of Highly Compressed GIF Data (Data Amplification).
  - CVE-2022-45199 - Pillow before 9.3.0 allows denial of service via SAMPLESPERPIXEL.

Chapter 47. PyTorch Release 23.03

The NVIDIA container image for PyTorch, release 23.03, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.8/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8
- NVIDIA CUDA® 12.1.0
- NVIDIA cuBLAS from CUDA 12.1.0
- NVIDIA cuDNN 8.8.1.3
- NVIDIA NCCL 2.17.1
- NVIDIA RAPIDS™ 23.02
- Apex
- rdma-core 36.0
- NVIDIA HPC-X 2.13
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2023.1.0.15
- Nsight Systems 2023.1.1.127
- NVIDIA TensorRT™ 8.5.3
- Torch-TensorRT 1.4.0dev0
- NVIDIA DALI® 1.23.0
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 0.6.0
- PyTorch quantization wheel 2.1.2

Driver Requirements

Release 23.03 is based on CUDA 12.1.0, which requires NVIDIA Driver release 530 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), 510.47 (or later R510), 515.65 (or later R515), 525.85 (or later R525), or 530.30 (or later R530).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R460, and R520 drivers, which are not forward-compatible with CUDA 12.1. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 23.03 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, NVIDIA Ampere architecture, and NVIDIA Hopper™ architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 23.03 is based on 2.0.0a0+1767026.

Announcements

- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- NVIDIA Deep Learning Profiler (DLProf) v1.8, which was included in the 21.12 container, was the final release of DLProf.

  Starting with the 22.01 container, DLProf is no longer included, but it can still be manually installed by using a pip wheel on the `nvidia-pyindex`.
- A preview of Torch-TensorRT (1.4.0dev0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container.

  To profile models in PyTorch, use DLProf.

  DLProf can help data scientists, engineers, and researchers understand improve performance of their models with visualization by using the DLProf Viewer in a web browser or by analyzing text reports. DL Prof is available on NGC or through a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`).

  You can obtain the models from Github or the NVIDIA GPU Cloud (NGC) instead. Some Python packages that were included in previous containers to support these example models have also been removed. Depending on their specific use cases, you might need to add some packages that were previously preinstalled.
- Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.8/site-packages"` if your Conda package manager was installed in `/opt/conda`.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.03	20.04	NVIDIA CUDA 12.1.0	2.0.0a0+1767026	TensorRT 8.5.3
23.02		NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation and a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.

- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.
- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.

- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.

- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- The following CVEs might be flaggted but were patched by backporting the fixes into the corresponding libraries in our release:
  - CVE-2022-45198 - Pillow before 9.2.0 performs Improper Handling of Highly Compressed GIF Data (Data Amplification).
  - CVE-2022-45199 - Pillow before 9.3.0 allows denial of service via SAMPLESPERPIXEL.
- Vgg16 training use cases might see a large performance regression on V100 when the workload is using close to all available device memory due to an unexpected memory thrashing when `` `torch.backends.cudnn.benchmark = True` `` is used. The performance can be restored by disabling `` `cudnn.benchmark` `` or by reducing the memory usage.

Chapter 48. PyTorch Release 23.02

The NVIDIA container image for PyTorch, release 23.02, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.8/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8
- NVIDIA CUDA® 12.0.1
- NVIDIA cuBLAS from CUDA 12.0.1
- NVIDIA cuDNN 8.7.0.84
- NVIDIA NCCL 2.16.5 (optimized for NVIDIA NVLink®)
- NVIDIA RAPIDS™ 22.12.0
- Apex
- rdma-core 36.0
- NVIDIA HPC-X 2.13
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2022.4.1.6
- Nsight Systems 2022.5.1
- NVIDIA TensorRT™ 8.5.3
- Torch-TensorRT 1.4.0dev0
- NVIDIA DALI® 1.22.0
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 0.5

Driver Requirements

Release 23.02 is based on CUDA 12.0.1, which requires NVIDIA Driver release 525 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), 510.47 (or later R510), 515.65 (or later R515), or 525.85 (or later R525).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, R460, and R520 drivers, which are not forward-compatible with CUDA 12.0. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 23.02 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, NVIDIA Ampere architecture, and NVIDIA Hopper™ architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 23.02 is based on 1.14.0a0+44dac51.

Announcements

- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- NVIDIA Deep Learning Profiler (DLProf) v1.8, which was included in the 21.12 container, was the final release of DLProf.

  Starting with the 22.01 container, DLProf is no longer included, but it can still be manually installed by using a pip wheel on the `nvidia-pyindex`.
- A preview of Torch-TensorRT (1.1.0a0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container.

  To profile models in PyTorch, use DLProf.

  DLProf can help data scientists, engineers, and researchers understand improve performance of their models with visualization by using the DLProf Viewer in a web browser or by analyzing text reports. DL Prof is available on NGC or through a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`).

  You can obtain the models from Github or the NVIDIA GPU Cloud (NGC) instead. Some Python packages that were included in previous containers to support these example models have also been removed. Depending on their specific use cases, you might need to add some packages that were previously preinstalled.
- Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.8/site-packages"` if your Conda package manager was installed in `/opt/conda`.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.02	20.04	NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	TensorRT 8.5.3
23.01				TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation and a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.

- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.
- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.

- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.

- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- The following CVEs might be flaggted but were patched by backporting the fixes into the corresponding libraries in our release:
  - CVE-2022-45198 - Pillow before 9.2.0 performs Improper Handling of Highly Compressed GIF Data (Data Amplification).
  - CVE-2022-45199 - Pillow before 9.3.0 allows denial of service via SAMPLESPERPIXEL.
  - CVE-2021-29063 - Mpmath v1.0.0 through v1.2.1 exposes a Regular Expression Denial of Service (ReDOS) vulnerability.
- Known security vulnerabilities:
  - CVE-2022-32212, CVE-2022-43548, CVE-2023-0286, CVE-2022-32223, CVE-2023-0286, CVE-2022-25881, CVE-2022-35255 for nodejs and openssl

Chapter 49. PyTorch Release 23.01

The NVIDIA container image for PyTorch, release 23.01, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.8/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8
- NVIDIA CUDA® 12.0.1
- NVIDIA cuBLAS from CUDA 12.0.1
- NVIDIA cuDNN 8.7.0.84
- NVIDIA NCCL 2.16.5 (optimized for NVIDIA NVLink®)
- NVIDIA RAPIDS™ 22.12.0 (For x86, only these libraries are included: cudf, xgboost, rmm, cuml, and cugraph.)
- Apex
- rdma-core 36.0
- NVIDIA HPC-X 2.13
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2022.4.1.6
- Nsight Systems 2022.5.1
- NVIDIA TensorRT™ 8.5.2.2
- Torch-TensorRT 1.4.0dev0
- NVIDIA DALI® 1.21.0
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 0.4

Driver Requirements

Release 23.01 is based on CUDA 12.0.1, which requires NVIDIA Driver release 525 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), 510.47 (or later R510), 515.65 (or later R515), or 525.85 (or later R525).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, and R460 drivers, which are not forward-compatible with CUDA 12.0. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 23.01 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, NVIDIA Ampere architecture, and NVIDIA Hopper™ architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 23.01 is based on 1.14.0a0+44dac51.

Announcements

- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- NVIDIA Deep Learning Profiler (DLProf) v1.8, which was included in the 21.12 container, was the final release of DLProf.

  Starting with the 22.01 container, DLProf is no longer included, but it can still be manually installed by using a pip wheel on the `nvidia-pyindex`.
- A preview of Torch-TensorRT (1.1.0a0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container.

  To profile models in PyTorch, use DLProf.

  DLProf can help data scientists, engineers, and researchers understand improve performance of their models with visualization by using the DLProf Viewer in a web browser or by analyzing text reports. DL Prof is available on NGC or through a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`).

  You can obtain the models from Github or the NVIDIA GPU Cloud (NGC) instead. Some Python packages that were included in previous containers to support these example models have also been removed. Depending on their specific use cases, you might need to add some packages that were previously preinstalled.
- Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.8/site-packages"` if your Conda package manager was installed in `/opt/conda`.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
23.01	20.04	NVIDIA CUDA 12.0.1	1.14.0a0+44dac51	TensorRT 8.5.2.2
22.12		NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation and a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.

- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.
- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.

- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.

- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- Certain cuDNN cases that use runtime compilation via NVRTC, particularly on ARM SBSA systems, can fail with `CUDNN_STATUS_RUNTIME_PREREQUISITE_MISSING`. A workaround for this situation is to export `LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda-11/lib64` . This will be fixed in an upcoming release.
- GNMTv2 performance regression of up to 50% for inference use cases and up to 20% for training.
- The following CVEs might be flaggted but were patched by backporting the fixes into the corresponding libraries in our release:
  - CVE-2022-45198 - Pillow before 9.2.0 performs Improper Handling of Highly Compressed GIF Data (Data Amplification).
  - CVE-2022-45199 - Pillow before 9.3.0 allows denial of service via SAMPLESPERPIXEL.
  - CVE-2021-29063 - Mpmath v1.0.0 through v1.2.1 exposes a Regular Expression Denial of Service (ReDOS) vulnerability.
- Known security vulnerabilities:
  - CVE-2022-25882 for ONNX<1.13.0

Chapter 50. PyTorch Release 22.12

The NVIDIA container image for PyTorch, release 22.12, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.8/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8
- NVIDIA CUDA® 11.8.0
- NVIDIA cuBLAS 11.11.3.6
- NVIDIA cuDNN 8.7.0.84
- NVIDIA NCCL 2.15.5 (optimized for NVIDIA NVLink®)
- NVIDIA RAPIDS™ 22.10.01 (For x86, only these libraries are included: cudf, xgboost, rmm, cuml, and cugraph.)
- Apex
- rdma-core 36.0
- NVIDIA HPC-X 2.13
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2022.3.0.0
- Nsight Systems 2022.4.2.1
- NVIDIA TensorRT™ 8.5.1
- Torch-TensorRT 1.1.0a0
- NVIDIA DALI® 1.20.0
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 0.3.0

Driver Requirements

Release 22.12 is based on CUDA 11.8.0, which requires NVIDIA Driver release 520 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), 510.47 (or later R510), or 515.65 (or later R515).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, and R460 drivers, which are not forward-compatible with CUDA 11.8. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 22.12 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, NVIDIA Ampere architecture, and NVIDIA Hopper™ architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 22.12 is based on 1.14.0a0+410ce96.

Announcements

- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- NVIDIA Deep Learning Profiler (DLProf) v1.8, which was included in the 21.12 container, was the final release of DLProf.

  Starting with the 22.01 container, DLProf is no longer included, but it can still be manually installed by using a pip wheel on the `nvidia-pyindex`.
- A preview of Torch-TensorRT (1.1.0a0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container.

  To profile models in PyTorch, use DLProf.

  DLProf can help data scientists, engineers, and researchers understand improve performance of their models with visualization by using the DLProf Viewer in a web browser or by analyzing text reports. DL Prof is available on NGC or through a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`).

  You can obtain the models from Github or the NVIDIA GPU Cloud (NGC) instead. Some Python packages that were included in previous containers to support these example models have also been removed. Depending on their specific use cases, you might need to add some packages that were previously preinstalled.
- Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.8/site-packages"` if your Conda package manager was installed in `/opt/conda`.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.12	20.04	NVIDIA CUDA 11.8.0	1.14.0a0+410ce96	TensorRT 8.5.1
22.11			1.13.0a0+936e930	
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation and a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.
- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized

version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.

- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.

- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

This model script is available on GitHub and NGC.

Known Issues

- GNMTv2 inference performance regression of up to 50% due to an MKL slowdown. Other CPU-sensitive workloads could also be affected.
- The following CVEs might be flaggted but were patched by backporting the fixes into the corresponding libraries in our release:
  - CVE-2022-45198 - Pillow before 9.2.0 performs Improper Handling of Highly Compressed GIF Data (Data Amplification).
  - CVE-2022-45199 - Pillow before 9.3.0 allows denial of service via SAMPLESPERPIXEL.
  - CVE-2021-29063 - Mpmath v1.0.0 through v1.2.1 exposes a Regular Expression Denial of Service (ReDOS) vulnerability.

Chapter 51. PyTorch Release 22.11

The NVIDIA container image for PyTorch, release 22.11, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the default Python environment (`/usr/local/lib/python3.8/dist-packages/torch`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8
- NVIDIA CUDA® 11.8.0
- NVIDIA cuBLAS 11.11.3.6
- NVIDIA cuDNN 8.7.0
- NVIDIA NCCL 2.15.5 (optimized for NVIDIA NVLink®)
- NVIDIA RAPIDS™ 22.10 (For x86, only these libraries are included: cudf, xgboost, rmm, cuml, and cugraph.)
- Apex
- rdma-core 36.0
- NVIDIA HPC-X 2.12.2tp1
- OpenMPI 4.1.4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2022.3.0.0
- Nsight Systems 2022.4.2.1
- NVIDIA TensorRT™ 8.5.1
- Torch-TensorRT 1.1.0a0
- NVIDIA DALI® 1.18.0
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 0.2.0

Driver Requirements

Release 22.11 is based on CUDA 11.8.0, which requires NVIDIA Driver release 520 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), 510.47 (or later R510), or 515.65 (or later R515).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, and R460 drivers, which are not forward-compatible with CUDA 11.8. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 22.11 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, NVIDIA Ampere architecture, and NVIDIA Hopper™ architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 22.11 is based on 1.13.0a0+936e930.

Announcements

- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- NVIDIA Deep Learning Profiler (DLProf) v1.8, which was included in the 21.12 container, was the final release of DLProf.

  Starting with the 22.01 container, DLProf is no longer included, but it can still be manually installed by using a pip wheel on the `nvidia-pyindex`.
- A preview of Torch-TensorRT (1.1.0a0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container.

  To profile models in PyTorch, use DLProf.

  DLProf can help data scientists, engineers, and researchers understand improve performance of their models with visualization by using the DLProf Viewer in a web browser or by analyzing text reports. DL Prof is available on NGC or through a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`).

  You can obtain the models from Github or the NVIDIA GPU Cloud (NGC) instead. Some Python packages that were included in previous containers to support these example models have also been removed. Depending on their specific use cases, you might need to add some packages that were previously preinstalled.
- Starting with the 22.11 PyTorch NGC container, miniforge is removed and all Python packages are installed in the default Python environment. In case you depend on Conda-specific packages, which might not be available on PyPI, we recommend building these packages from source. A workaround is to manually install a Conda package manager, and add the conda path to your PYTHONPATH for example, using `export PYTHONPATH="/opt/conda/lib/python3.8/site-packages"` if your Conda package manager was installed in `/opt/conda`.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.11	20.04	NVIDIA CUDA 11.8.0	1.13.0a0+936e930	TensorRT 8.5.1
22.10			1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation and a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.
- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized

version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.

- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.

- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

This model script is available on GitHub and NGC.

Known Issues

- In rare cases, cuDNN autotuning could cause a long startup time or a hang. In these cases, disbale autotuning using `torch.backends.cudnn.benchmark = False`.
- GNMTv2 inference performance regression of up to 50% due to an MKL slowdown. Other CPU-sensitive workloads could also be affected.
- BERT pretraining performance regression of up to 17% for workloads using dynamic input shapes.
- Tacotron2 inference performance regression of up to 15% for workloads using dynamic input shapes.

Security CVEs

- CVE-2022-45198 - Pillow before 9.2.0 performs Improper Handling of Highly Compressed GIF Data (Data Amplification).
- CVE-2022-45199 - Pillow before 9.3.0 allows denial of service via SAMPLESPERPIXEL.

Chapter 52. PyTorch Release 22.10

The NVIDIA container image for PyTorch, release 22.10, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8
- NVIDIA CUDA® 11.8.0
- NVIDIA cuBLAS 11.11.3.6
- NVIDIA cuDNN 8.6.0.163
- NVIDIA NCCL 2.15.5 (optimized for NVIDIA NVLink®)
- NVIDIA RAPIDS™ 22.08.01 (For x86, only these libraries are included: cudf, xgboost, rmm, cuml, and cugraph.)
- Apex
- rdma-core 36.0
- NVIDIA HPC-X 2.12.2tp1
- OpenMPI 4.1.5a1
- GDRCopy 2.3
- TensorBoard 2.10.0
- Nsight Compute 2022.3.0.22
- Nsight Systems 2022.4.2.1
- NVIDIA TensorRT™ 8.5.0.12
- Torch-TensorRT 1.1.0a0
- NVIDIA DALI® 1.18.0
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 0.1.0

Driver Requirements

Release 22.10 is based on CUDA 11.8.0, which requires NVIDIA Driver release 520 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), 510.47 (or later R510), or 515.65 (or later R515).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, and R460 drivers, which are not forward-compatible with CUDA 11.8. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 22.10 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, NVIDIA Ampere architecture, and NVIDIA Hopper™ architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 22.10 is based on 1.13.0a0+d0d6b1f.

Announcements

- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- NVIDIA Deep Learning Profiler (DLProf) v1.8, which was included in the 21.12 container, was the final release of DLProf.

  Starting with the 22.01 container, DLProf is no longer included, but it can still be manually installed by using a pip wheel on the `nvidia-pyindex`.
- A preview of Torch-TensorRT (1.1.0a0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container.

  To profile models in PyTorch, use DLProf.

  DLProf can help data scientists, engineers, and researchers understand improve performance of their models with visualization by using the DLProf Viewer in a web browser or by analyzing text reports. DL Prof is available on NGC or through a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`).

  You can obtain the models from Github or the NVIDIA GPU Cloud (NGC) instead. Some Python packages that were included in previous containers to support these example models have also been removed. Depending on their specific use cases, you might need to add some packages that were previously preinstalled.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.10	20.04	NVIDIA CUDA 11.8.0	1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.09				
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation and a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.

- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.

- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.

- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.

- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:
  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.
- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.
- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.
- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.
- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- None.

Chapter 53. PyTorch Release 22.09

The NVIDIA container image for PyTorch, release 22.09, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8
- NVIDIA CUDA® 11.8.0
- NVIDIA cuBLAS 11.11.3.6
- NVIDIA cuDNN 8.6.0.163
- NVIDIA NCCL 2.15.1 (optimized for NVIDIA NVLink®)
- NVIDIA RAPIDS™ 22.08.01 (For x86, only these libraries are included: cudf, xgboost, rmm, cuml, and cugraph.)
- Apex
- rdma-core 36.0
- NVIDIA HPC-X 2.12.1a0
- OpenMPI 4.1.2rc4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2022.3.0.22
- Nsight Systems 2022.3.1.43
- NVIDIA TensorRT™ 8.5.0.12
- Torch-TensorRT 1.1.0a0
- NVIDIA DALI® 1.17.0
- MAGMA 2.6.2
- JupyterLab 2.3.2 including Jupyter-TensorBoard
- TransformerEngine 0.1.0

Driver Requirements

Release 22.09 is based on CUDA 11.8.0, which requires NVIDIA Driver release 520 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), 510.47 (or later R510), or 515.65 (or later R515).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, and R460 drivers, which are not forward-compatible with CUDA 11.8. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 22.09 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, NVIDIA Ampere architecture, and NVIDIA Hopper™ architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 22.09 is based on 1.13.0a0+d0d6b1f.
- CUDA lazy module loading is on by default. To disable it, use `unset CUDA_MODULE_LOADING` or set it to `EAGER`.
- The experimental cuDNN v8 API is enabled by default. To disable it, use `` `unset TORCH_CUDNN_V8_API_ENABLED` `` or set it to `` `0` ``.
- TransformerEngine v0.1.0 to support FP8 on Hopper.

Announcements

- Transformer Engine is a library for accelerating Transformer models on NVIDIA GPUs. It includes support for 8-bit floating point (FP8) precision on Hopper GPUs which provides better training and inference performance with lower memory utilization. Transformer Engine also includes a collection of highly optimized modules for popular Transformer architectures and an automatic mixed precision-like API that can be used seamlessly with your PyTorch code.
- NVIDIA Deep Learning Profiler (DLProf) v1.8, which was included in the 21.12 container, was the final release of DLProf.

  Starting with the 22.01 container, DLProf is no longer included, but it can still be manually installed by using a pip wheel on the `nvidia-pyindex`.
- A preview of Torch-TensorRT (1.1.0a0) is now included.

Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.

- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container.

  To profile models in PyTorch, use DLProf.

  DLProf can help data scientists, engineers, and researchers understand improve performance of their models with visualization by using the DLProf Viewer in a web browser or by analyzing text reports. DL Prof is available on NGC or through a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`).

  You can obtain the models from Github or the NVIDIA GPU Cloud (NGC) instead. Some Python packages that were included in previous containers to support these example models have also been removed. Depending on their specific use cases, you might need to add some packages that were previously preinstalled.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.09	20.04	NVIDIA CUDA 11.8.0	1.13.0a0+d0d6b1f	TensorRT 8.5.0.12
22.08		NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation and a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores.

This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.
- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:
  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.
- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.
- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.
- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.
- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- Up to 49% performance regression in DLRM on multi-GPU training runs.

- On H100 NVLink systems using 2 GPUs for training, certain communication patterns can trigger a corner-case bug that manifests either as a hang or as an "illegal instruction" exception. A workaround for this case is to set the environment variable `NCCL_PROTO=^LL128`. This issue will be addressed in an upcoming release.

Chapter 54. PyTorch Release 22.08

The NVIDIA container image for PyTorch, release 22.08, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8
- NVIDIA CUDA® 11.7.1
- NVIDIA cuBLAS 11.10.3.66
- NVIDIA cuDNN 8.5.0.96
- NVIDIA NCCL 2.12.12 (optimized for NVIDIA NVLink®)
- NVIDIA RAPIDS™ 22.06 (For x86, only these libraries are included: cudf, xgboost, rmm, cuml, and cugraph.)
- Apex
- rdma-core 36.0
- NVIDIA HPC-X 2.10 with UCX 1.12.0
- OpenMPI 4.1.2rc4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2022.2.1
- Nsight Systems 2022.1.3.18
- NVIDIA TensorRT™ 8.4.2.4
- Torch-TensorRT 1.1.0a0
- NVIDIA DALI® 1.16.0
- MAGMA 2.6.2
- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0

- Jupyter Core 4.6.1
- Jupyter Notebook 6.0.3
- JupyterLab 2.3.2, including Jupyter-TensorBoard
- JupyterLab Server 1.0.6
- Jupyter-TensorBoard

Driver Requirements

Release 22.08 is based on CUDA 11.7.1, which requires NVIDIA Driver release 515 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), or 510.47 (or later R510).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, and R460 drivers, which are not forward-compatible with CUDA 11.7. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 22.08 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, and NVIDIA Ampere Architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 22.08 is based on 1.13.0a0+d321be6.
- CUDA Module loading is set to LAZY starting with the 22.08 container. To enable the default eager loading behavior, use `` `export CUDA_MODULE_LOADING=EAGER` `` or `` `unset CUDA_MODULE_LOADING` ``. Refer to the CUDA C++ Programming Guide for more information about this environment variable.

Announcements

- NVIDIA Deep Learning Profiler (DLProf) v1.8, which was included in the 21.12 container, was the final release of DLProf.

  Starting with the 22.01 container, DLProf is no longer included, but it can still be manually installed by using a pip wheel on the `nvidia-pyindex`.
- A preview of Torch-TensorRT (1.1.0a0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.

- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container.

  To profile models in PyTorch, use DLProf.

  DLProf can help data scientists, engineers, and researchers understand improve performance of their models with visualization by using the DLProf Viewer in a web browser or by analyzing text reports. DL Prof is available on NGC or through a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`).

  You can obtain the models from Github or the NVIDIA GPU Cloud (NGC) instead. Some Python packages that were included in previous containers to support these example models have also been removed. Depending on their specific use cases, you might need to add some packages that were previously preinstalled.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.08	20.04	NVIDIA CUDA 11.7.1	1.13.0a0+d321be6	TensorRT 8.4.2.4
22.07		NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation and a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only three lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

APEX AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over APEX AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- APEX AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.

- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.

- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.

- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.

- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. The NVIDIA BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:
  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.
- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.
- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.
- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.
- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- Performance regression on Volta and NVIDIA Ampere Architecture for ResNet-like model inference use cases of up to 18%.
- Performance regression on Volta and NVIDIA Ampere Architecture for Tactron2+Waveglow inference use cases of up to 35%.
- The default `antialiasing` argument for resizing operations in DALI 1.16.0 was changed to `True`, which could result in performance regressions on CPU-limited use cases. Set this argument to `False` to restore the previous behavior.

Chapter 55. PyTorch Release 22.07

The NVIDIA container image for PyTorch, release 22.07, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8
- NVIDIA CUDA® 11.7 Update 1 Preview
- NVIDIA cuBLAS 11.10.3.66
- NVIDIA cuDNN 8.4.1
- NVIDIA NCCL 2.12.12 (optimized for NVIDIA NVLink®)
- NVIDIA RAPIDS™ 22.04 (For x86, only these libraries are included: cudf, xgboost, rmm, cuml, and cugraph.)
- Apex
- rdma-core 36.0
- NVIDIA HPC-X 2.10 with UCX 1.12.0
- OpenMPI 4.1.2rc4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2022.2.1
- Nsight Systems 2022.1.3.3
- NVIDIA TensorRT™ 8.4.1
- Torch-TensorRT 1.1.0a0
- NVIDIA DALI® 1.15.0
- MAGMA 2.6.2
- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0

- Jupyter Core 4.6.1
- Jupyter Notebook 6.0.3
- JupyterLab 2.3.2, including Jupyter-TensorBoard
- JupyterLab Server 1.0.6
- Jupyter-TensorBoard

Driver Requirements

Release 22.07 is based on CUDA 11.7 Update 1 Preview, which requires NVIDIA Driver release 515 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), or 510.47 (or later R510).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, and R460 drivers, which are not forward-compatible with CUDA 11.7. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 22.07 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, and NVIDIA Ampere GPU architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 22.07 is based on 1.13.0a0+08820cb.

Announcements

- NVIDIA Deep Learning Profiler (DLProf) v1.8, which was included in the 21.12 container, was the final release of DLProf.

  Starting with the 22.01 container, DLProf is no longer included, but it can still be manually installed by using a pip wheel on the `nvidia-pyindex`.
- A preview of Torch-TensorRT (1.1.0a0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container.

  To profile models in PyTorch, use DLProf.

  DLProf can help data scientists, engineers, andand researchers underst and improve performance of their models with visualization by using the DLProf Viewer in a web browser or by analyzing text reports. DL Prof is available on NGC or through a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`).

  You can obtain the models from Github or the NVIDIA GPU Cloud (NGC) instead. Some Python packages that were included in previous containers to support these example models have also been removed. Depending on their specific use cases, you might need to add some packages that were previously preinstalled.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.07	20.04	NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+08820cb	TensorRT 8.4.1
22.06			1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream	TensorRT 5.1.5

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
			commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation and a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over Apex AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.

- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.
- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. NVIDIA's BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

- A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
- A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.
- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.
- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.
- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

None.

Chapter 56. PyTorch Release 22.06

The NVIDIA container image for PyTorch, release 22.06, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8
- NVIDIA CUDA® 11.7 Update 1 Preview
- NVIDIA cuBLAS 11.10.3.66
- NVIDIA cuDNN 8.4.1
- NVIDIA NCCL 2.12.10 (optimized for NVIDIA NVLink®)
- NVIDIA RAPIDS™ 22.04 (For x86, only these libraries are included: cudf, xgboost, rmm, cuml, and cugraph.)
- Apex
- rdma-core 36.0
- NVIDIA HPC-X 2.10 with UCX 1.12.0
- OpenMPI 4.1.2rc4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2022.2.1
- Nsight Systems 2022.1.3.3
- NVIDIA TensorRT™ 8.2.5
- Torch-TensorRT 1.1.0a0
- NVIDIA DALI® 1.14.0
- MAGMA 2.6.2
- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0

- Jupyter Core 4.6.1
- Jupyter Notebook 6.0.3
- JupyterLab 2.3.2, including Jupyter-TensorBoard
- JupyterLab Server 1.0.6
- Jupyter-TensorBoard

Driver Requirements

Release 22.06 is based on CUDA 11.7 Update 1 Preview, which requires NVIDIA Driver release 515 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), or 510.47 (or later R510).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, and R460 drivers, which are not forward-compatible with CUDA 11.7. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 22.06 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, and NVIDIA Ampere GPU architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 22.06 is based on 1.13.0a0+340c412.
- The TF32 numerical format is enabled by default for cuBLAS and cuDNN operations starting with the 22.06 release. If you encounter training issues especially for regression, generative or higher-order models, or by using TF32 operations in pre- or post-processing steps, try to disable TF32 by setting `` `torch.set_float32_matmul_precision('highest')` ``.

Announcements

- NVIDIA Deep Learning Profiler (DLProf) v1.8, which was included in the 21.12 container, was the final release of DLProf.

  Starting with the 22.01 container, DLProf is no longer included, but it can still be manually installed by using a pip wheel on the `nvidia-pyindex`.
- A preview of Torch-TensorRT (1.1.0a0) is now included.

Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.

- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container.

  To profile models in PyTorch, use DLProf.

  DLProf can help data scientists, engineers, andand researchers underst and improve performance of their models with visualization by using the DLProf Viewer in a web browser or by analyzing text reports. DL Prof is available on NGC or through a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`).

  You can obtain the models from Github or the NVIDIA GPU Cloud (NGC) instead. Some Python packages that were included in previous containers to support these example models have also been removed. Depending on their specific use cases, you might need to add some packages that were previously preinstalled.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.06	20.04	NVIDIA CUDA 11.7 Update 1 Preview	1.13.0a0+340c412	TensorRT 8.2.5
22.05		NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation and a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over Apex AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.

- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.

- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.

- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.

- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. NVIDIA's BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:
  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.
- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.
- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.
- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.
- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

None.

Chapter 57. PyTorch Release 22.05

The NVIDIA container image for PyTorch, release 22.05, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8
- NVIDIA CUDA® 11.7.0
- NVIDIA cuBLAS 11.10.1.25
- NVIDIA cuDNN 8.4.0.27
- NVIDIA NCCL 2.12.10 (optimized for NVIDIA NVLink®)
- NVIDIA RAPIDS™ 22.04 (For x86, only these libraries are included: cudf, xgboost, rmm, cuml, and cugraph.)
- Apex
- rdma-core 36.0
- NVIDIA HPC-X 2.10 with UCX 1.12.0
- OpenMPI 4.1.2rc4+
- GDRCopy 2.3
- TensorBoard 2.9.0
- Nsight Compute 2022.2.0.13
- Nsight Systems 2022.1.3.3
- NVIDIA TensorRT™ 8.2.5.1
- Torch-TensorRT 1.1.0a0
- NVIDIA DALI® 1.13.0
- MAGMA 2.6.2
- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0

- Jupyter Core 4.6.1
- Jupyter Notebook 6.0.3
- JupyterLab 2.3.2, including Jupyter-TensorBoard
- JupyterLab Server 1.0.6
- Jupyter-TensorBoard

Driver Requirements

Release 22.05 is based on CUDA 11.7, which requires NVIDIA Driver release 515 or later. However, if you are running on a data center GPU (for example, T4 or any other data center GPU), you can use NVIDIA driver release 450.51 (or later R450), 470.57 (or later R470), or 510.47 (or later R510).

The CUDA driver's compatibility package only supports particular drivers. Thus, users should upgrade from all R418, R440, and R460 drivers, which are not forward-compatible with CUDA 11.7. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 22.05 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, and NVIDIA Ampere GPU architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 22.05 is based on 1.12.0a0+8a1a93a.
- CUDA 11.7 introduces lazy module loading, which can save memory usage in your application. To enable it, use `export CUDA_MODULE_LOADING="LAZY"`.

Announcements

- NVIDIA Deep Learning Profiler (DLProf) v1.8, which was included in the 21.12 container, was the final release of DLProf.

  Starting with the 22.01 container, DLProf is no longer included, but it can still be manually installed by using a pip wheel on the `nvidia-pyindex`.
- A preview of Torch-TensorRT (1.1.0a0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.

- Starting with the 22.05 release, the PyTorch container is available for the Arm SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container.

  To profile models in PyTorch, use DLProf.

  DLProf can help data scientists, engineers, andand researchers underst and improve performance of their models with visualization by using the DLProf Viewer in a web browser or by analyzing text reports. DL Prof is available on NGC or through a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`).

  You can obtain the models from Github or the NVIDIA GPU Cloud (NGC) instead. Some Python packages that were included in previous containers to support these example models have also been removed. Depending on their specific use cases, you might need to add some packages that were previously preinstalled.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.05	20.04	NVIDIA CUDA 11.7.0	1.12.0a0+8a1a93a	TensorRT 8.2.5.1
22.04		NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for Arm SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream	TensorRT 5.1.5

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
			commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation and a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over Apex AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on NVIDIA Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.

- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

  This model script is available on GitHub.
- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.
- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. NVIDIA's BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.
- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

- A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
- A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

  The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

  This model script is available on GitHub and NGC.
- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.
- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.
- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

- Up to 30% inference and 21% training performance drop for EfficientDet-D0.
- Up to 20% inference and 17% training performance drop for NCF.
- Potential out-of-memory issues in Tacotron2 when modules are scripted in amp. Disable autocast in TorchScript by using `` `torch._C._jit_set_autocast_mode(False)` `` if you encounter this issue.

Chapter 58. PyTorch Release 22.04

The NVIDIA container image for PyTorch, release 22.04, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8
- NVIDIA CUDA® 11.6.2
- cuBLAS 11.9.3.115
- NVIDIA cuDNN 8.4.0.27
- NVIDIA NCCL 2.12.10 (optimized for NVIDIA NVLink®)
- NVIDIA RAPIDS™ 22.02 (For x86, only these libraries are included: cudf, xgboost, rmm, cuml, and cugraph.)
- Apex
- rdma-core 36.0
- NVIDIA HPC-X 2.10 with UCX 1.12.0
- OpenMPI 4.1.2rc4+
- GDRCopy 2.3
- TensorBoard 2.8.0
- Nsight Compute 2022.1.1.2
- Nsight Systems 2022.2.1.31-5fe97ab
- NVIDIA TensorRT™ 8.2.4.2
- Torch-TensorRT 1.1.0a0
- NVIDIA DALI® 1.12.0
- MAGMA 2.5.2
- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0

- Jupyter Core 4.6.1
- Jupyter Notebook 6.0.3
- JupyterLab 2.3.2, including Jupyter-TensorBoard
- JupyterLab Server 1.0.6
- Jupyter-TensorBoard

Driver Requirements

Release 22.04 is based on CUDA 11.6.2, which requires NVIDIA Driver release 510 or later. However, if you are running on a Data Center GPU (for example, T4 or any other Tesla board), use NVIDIA driver release 418.40 (or later R418), 440.33 (or later R440), 450.51 (or later R450), 460.27 (or later R460), or 470.57 (or later R470). The CUDA driver's compatibility package only supports specific drivers. For a complete list of supported drivers, see CUDA Application Compatibility. For more information, see CUDA Compatibility and Upgrades and NVIDIA CUDA and Drivers Support.

GPU Requirements

Release 22.04 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, and NVIDIA Ampere GPU architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 22.04 is based on 1.12.0a0+bd13bc6.

Announcements

- NVIDIA Deep Learning Profiler (DLProf) v1.8, which was included in the 21.12 container, was the final release of DLProf.

  Starting with the 22.01 container, DLProf is no longer included, but it can still be manually installed by using a pip wheel on the `nvidia-pyindex`.
- A preview of Torch-TensorRT (1.1.0a0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 21.10 release, a beta version of the PyTorch container is available for the ARM SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.

- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container.

  To profile models in PyTorch, use DLProf.

  DLProf can help data scientists, engineers and researchers understand and improve performance of their models with visualization by using the DLProf Viewer in a web browser or by analyzing text reports. DL Prof is available on NGC or through a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`).

  You can obtain the models from Github or the NVIDIA GPU Cloud (NGC) instead. Some Python packages that were included in previous containers to support these example models have also been removed. Depending on their specific use cases, you might need to add some packages that were previously preinstalled.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.04	20.04	NVIDIA CUDA 11.6.2	1.12.0a0+bd13bc6	TensorRT 8.2.4.2
22.03		NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for ARM SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation and a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over Apex AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments.

Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

This model script is available on GitHub.

- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.

- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. NVIDIA's BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA Tesla® V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

This model script is available on GitHub and NGC.

- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.

- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

Up to an 18% performance regression in the NCF inference on Volta GPUs.

Chapter 59. PyTorch Release 22.03

The NVIDIA container image for PyTorch, release 22.03, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and installed in the Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8 environment
- NVIDIA CUDA® 11.6.1
- cuBLAS 11.8.1.74
- NVIDIA cuDNN 8.3.3.40
- NVIDIA NCCL 2.12.9 (optimized for NVIDIA NVLink®™)
- NVIDIA RAPIDS™ 22.02 (For x86, only these libraries are included: cudf, xgboost, rmm, cuml, and cugraph.)
- Apex
- rdma-core 36.0
- NVIDIA HPC-X 2.10 with UCX 1.12.0
- OpenMPI 4.1.2rc4+
- GDRCopy 2.3
- TensorBoard 2.8.0
- Nsight Compute 2022.1.1.2
- Nsight Systems 2021.5.2.53
- NVIDIA TensorRT™ 8.2.3
- Torch-TensorRT 1.1.0a0
- NVIDIA DALI® 1.11.1
- MAGMA 2.5.2
- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0

- Jupyter Core 4.6.1
- Jupyter Notebook 6.0.3
- JupyterLab 2.3.2, including Jupyter-TensorBoard
- JupyterLab Server 1.0.6
- Jupyter-TensorBoard

Driver Requirements

Release 22.03 is based on CUDA 11.6.1, which requires NVIDIA Driver release 510 or later. However, if you are running on a Data Center GPU (for example, T4 or any other Tesla board), use NVIDIA driver release 418.40 (or later R418), 440.33 (or later R440), 450.51 (or later R450), 460.27 (or later R460), or 470.57 (or later R470). The CUDA driver's compatibility package only supports specific drivers. For a complete list of supported drivers, see CUDA Application Compatibility. For more information, see CUDA Compatibility and Upgrades and NVIDIA CUDA and Drivers Support.

GPU Requirements

Release 22.03 supports CUDA compute capability 6.0 and later. This corresponds to GPUs in the NVIDIA Pascal, NVIDIA Volta™, NVIDIA Turing™, and NVIDIA Ampere GPU architecture families. For a list of GPUs to which this compute capability corresponds, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 22.03 is based on 1.12.0a0+2c916ef.

Announcements

- NVIDIA Deep Learning Profiler (DLProf) v1.8, which was included in the 21.12 container, was the final release of DLProf.

  Starting with the 22.01 container, DLProf is no longer included, but it can still be manually installed by using a pip wheel on the `nvidia-pyindex`.
- A preview of Torch-TensorRT (1.1.0a0) is now included.

  Torch-TRT is the TensorRT integration for PyTorch and brings the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 21.10 release, a beta version of the PyTorch container is available for the ARM SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.

- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container.

  To profile models in PyTorch, use DLProf.

  DLProf can help data scientists, engineers and researchers understand and improve performance of their models with visualization by using the DLProf Viewer in a web browser or by analyzing text reports. DL Prof is available on NGC or through a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`).

  You can obtain the models from Github or the NVIDIA GPU Cloud (NGC) instead. Some Python packages that were included in previous containers to support these example models have also been removed. Depending on their specific use cases, you might need to add some packages that were previously preinstalled.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For earlier container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.03	20.04	NVIDIA CUDA 11.6.1	1.12.0a0+2c916ef	TensorRT 8.2.3
22.02		NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for ARM SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation and a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. AMP will select an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but `torch.cuda.amp` is the future-proof alternative and offers a number of advantages over Apex AMP.

- Guidance and examples demonstrating `torch.cuda.amp` can be found here.
- Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NGC focus on achieving the best performance and convergence from NVIDIA Volta™ tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and NVIDIA Turing™, so you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model: This model was introduced in the Aggregated Residual Transformations for Deep Neural Networks paper.

  It is based on the regular ResNet model, which substitutes 3x3 convolutions in the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model: This ResNeXt101-32x4d model has an added Squeeze-and-Excitation (SE) module that was introduced in the *Squeeze-and-Excitation Networks* paper.

  This model script is available on GitHub.
- TransformerXL model: This transformer-based language model has a segment-level recurrence and a novel relative positional encoding.

  The enhancements that were introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments.

Our implementation is based on the codebase that was published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters, our modifications were made to achieve better hardware usage and to take advantage of Tensor Cores.

This model script is available on GitHub.

- Jasper model: This repository provides an implementation of the Jasper model in PyTorch from the *Jasper: An End-to-End Convolutional Neural Acoustic Model* paper.

  The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without external data.

  This model script is available on GitHub and NGC.

- BERT model: Bidirectional Encoder Representations from Transformers (BERT) is a new method of pretraining language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

  This model is based on the *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. NVIDIA's BERT implementation is an optimized version of the Hugging Face implementation paper that leverages mixed-precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Mask R-CNN model: Mask R-CNN is a convolution-based neural network that is used for object instance segmentation.

  The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, which leverages mixed precision arithmetic by using Tensor Cores on NVIDIA Tesla® V100 GPUs for 1.3x faster training time while maintaining target accuracy.

  This model script is available on GitHub and NGC.

- Tacotron 2 and WaveGlow v1.1 model: This text-to-speech (TTS) system is a combination of the following neural network models:

  - A modified Tacotron 2 model from the *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions* paper
  - A flow-based neural network model from the *WaveGlow: A Flow-based Generative Network for Speech Synthesis* paper.

  This model script is available on GitHub and NGC.

- SSD300 v1.1 model: This model is based on the *SSD: Single Shot MultiBox Detector* paper.

The main difference between this model and the model described in the paper is in the backbone. Specifically, the VGG model is obsolete and has been replaced by the ResNet50 model.

This model script is available on GitHub and NGC.

- Neural Collaborative Filtering (NCF) model: This model focuses on providing recommendations, which is also known as collaborative filtering with implicit feedback.

  The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the *Neural Collaborative Filtering* paper.

  This model script is available on GitHub and NGC.

- ResNet50 v1.5 model: This model is a modified version of the original ResNet50 v1 model.

  This model script is available on GitHub and NGC.

- GNMT v2 model: This model is similar to the model that is discussed in the *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper.

  This model script is available on GitHub and NGC.

Known Issues

None

Chapter 60. PyTorch Release 22.02

The NVIDIA container image for PyTorch, release 22.02, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8 environment
- NVIDIA CUDA 11.6.0
- cuBLAS 11.8.1.74
- NVIDIA cuDNN 8.3.2.44
- NVIDIA NCCL 2.11.4 (optimized for NVLink™)
- RAPIDS 21.12 (For x86, only these libraries are included: cudf, xgboost, rmm, cuml, and cugraph.)
- APEX
- rdma-core 36.0
- NVIDIA HPC-X 2.10
- OpenMPI 4.1.2rc4+
- OpenUCX 1.12.0
- GDRCopy 2.3
- TensorBoard 2.8.0
- TensorRT 8.2.3
- Torch-TensorRT 1.1.0a0
- SHARP 2.5
- APEX
- Nsight Compute 2022.1.0.0
- Nsight Systems 2021.5.2.53
- TensorBoard 2.8.0

- DALI 1.10.0
- MAGMA 2.5.2
- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 2.3.2
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 22.02 is based on NVIDIA CUDA 11.6.0, which requires NVIDIA Driver release 510 or later. However, if you are running on a Data Center GPU (for example, T4 or any other Tesla board), you may use NVIDIA driver release 418.40 (or later R418), 440.33 (or later R440), 450.51 (or later R450), 460.27 (or later R460), or 470.57 (or later R470). The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades and NVIDIA CUDA and Drivers Support.

GPU Requirements

Release 22.02 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, Turing, and NVIDIA Ampere GPU architecture families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 22.02 is based on 1.11.0a0+17540c5c.

Announcements

- DLProf v1.8, which was included in the 21.12 container, was the last release of DLProf. Starting with the 22.01 container, DLProf is no longer included. It can still be manually installed via a pip wheel on the nvidia-pyindex.
- A preview of Torch-TensorRT (1.1.0a0) is now included. Torch-TRT is the TensorRT integration for PyTorch bringing the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 21.10 release, a beta version of the PyTorch container is available for the ARM SBSA platform.

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container. To profile models in PyTorch, please use NVIDIA Deep Learning Profiler (DLProf). DLProf can help data scientists, engineers and researchers understand and improve performance of their models with visualization via DLProf Viewer in the web browser, or by analyzing text reports. DL Prof is available on NGC or a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`). Instead they can be obtained from Github or the NVIDIA GPU Cloud (NGC). Some python packages, included in previous containers to support these example models, have also been removed. Depending on their specific use cases, users may need to add some packages that were previously pre-installed.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.02	20.04	NVIDIA CUDA 11.6.0	1.11.0a0+17540c5c	TensorRT 8.2.3
22.01			1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for ARM SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try

mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub
- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among

end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- Starting in 22.02 the PyTorch container does not build Caffe2 anymore. If scripted models were exported in the legacy format (using our 19.09 or previous NGC containers corresponding to PyTorch 1.2.0 or previous releases) you might need to re-export the model in order to be able to load it in 22.02.
- Torch-TensorRT `vgg_qat.ipynb notebook`:
  - There is a missing field `truncate_long_and_double=True` in torch_tensorrt compilation of a CIFAR10 based QAT model demonstrated in `vgg_qat.ipynb` (QAT notebook). Please add it to `vgg_qat.ipynb` before you compile the model with torch_tensorrt. The notebook can be found at `/opt/pytorch/torch_tensorrt/notebooks/vgg_qat.ipynb`.
  - The same field is also required for the post-training quantization sample to pass successfully. Please add `truncate_long_and_double=True` to the compile spec during torch_tensorrt compilation. Location : `/opt/pytorch/torch_tensorrt/examples/int8/ptq/main.cpp`.
- ARM
  - Passing external CUDA Streams to PyTorch via `torch.cuda.streams.ExternalStream(stream_v)` might fail and is being debugged.

Chapter 61. PyTorch Release 22.01

The NVIDIA container image for PyTorch, release 22.01, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8 environment
- NVIDIA CUDA 11.6.0
- cuBLAS 11.8.1.74
- NVIDIA cuDNN 8.3.2.44
- NVIDIA NCCL 2.11.4 (optimized for NVLink™)
- RAPIDS 21.12 (For x86, only these libraries are included: cudf, xgboost, rmm, cuml, and cugraph.)
- rdma-core 36.0
- NVIDIA HPC-X 2.10
- OpenMPI 4.1.2rc4+
- OpenUCX 1.12.0
- GDRCopy 2.3
- TensorRT 8.2.2
- Torch-TensorRT 1.1.0a0
- SHARP 2.5
- APEX
- Nsight Compute 2022.1.0.0
- Nsight Systems 2021.3.2.4
- TensorBoard 2.8.0
- DALI 1.9
- MAGMA 2.5.2

- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 2.3.2
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 22.01 is based on NVIDIA CUDA 11.6.0, which requires NVIDIA Driver release 510 or later. However, if you are running on a Data Center GPU (for example, T4 or any other Tesla board), you may use NVIDIA driver release 418.40 (or later R418), 440.33 (or later R440), 450.51 (or later R450), 460.27 (or later R460), or 470.57 (or later R470). The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades and NVIDIA CUDA and Drivers Support.

GPU Requirements

Release 22.01 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, Turing, and NVIDIA Ampere GPU architecture families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 22.01 is based on 1.11.0a0+bfe5ad28.
- The 22.01 container ships with a preview of the cuDNN v8 API with `bfloat16` support and can be enabled using `export CUDNN_V8_API_ENABLED=1`. To use the new neural network-based heuristics, use `export USE_HEURISTIC_MODE_B=1` in addition to `export CUDNN_V8_API_ENABLED=1`. Please refer to the cuDNN API docs for more information about this heuristic mode (https://docs.nvidia.com/deeplearning/cudnn/api/index.html).

Announcements

- DLProf v1.8, which was included in the 21.12 container, was the last release of DLProf. Starting with the 22.01 container, DLProf is no longer included. It can still be manually installed via a pip wheel on the nvidia-pyindex.

- A preview of Torch-TensorRT (1.1.0a0) is now included. Torch-TRT is the TensorRT integration for PyTorch bringing the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 21.10 release, a beta version of the PyTorch container is available for the ARM SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container. To profile models in PyTorch, please use NVIDIA Deep Learning Profiler (DLProf). DLProf can help data scientists, engineers and researchers understand and improve performance of their models with visualization via DLProf Viewer in the web browser, or by analyzing text reports. DL Prof is available on NGC or a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`). Instead they can be obtained from Github or the NVIDIA GPU Cloud (NGC). Some python packages, included in previous containers to support these example models, have also been removed. Depending on their specific use cases, users may need to add some packages that were previously pre-installed.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
22.01	20.04	NVIDIA CUDA 11.6.0	1.11.0a0+bfe5ad28	TensorRT 8.2.2
21.12		NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for ARM SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as	TensorRT 5.1.5

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
			well as a cherry-picked	

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is

based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub

- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- The version of OpenUCX included with PyTorch container image version 21.11 has known issues with RAPIDS UCX-Py. When using Dask with this container version, pass `protocol="tcp"` to `LocalCUDACluster()`, not `protocol="ucx"`, to work around these issues. Additionally, LocalCUDACluster UCX-specific configurations must remain unspecified; they are: `enable_tcp_over_ucx`, `enable_nvlink`, `enable_infiniband`, `enable_rdmacm` and `ucx_net_devices`.
- ARM
  - Passing external CUDA Streams to PyTorch via `` `torch.cuda.streams.ExternalStream(stream_v)` `` might fail and is being debugged.

Chapter 62. PyTorch Release 21.12

The NVIDIA container image for PyTorch, release 21.12, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8 environment
- NVIDIA CUDA 11.5.0
- cuBLAS 11.7.3.1
- NVIDIA cuDNN 8.3.1.22
- NVIDIA NCCL 2.11.4 (optimized for NVLink™)
- RAPIDS 21.10
- rdma-core 36.0
- OpenMPI 4.1.2a1
- OpenUCX 1.11.0rc1
- GDRCopy 2.3
- NVIDIA HPC-X 2.9
- TensorRT 8.2.1.8
- Torch-TensorRT 1.1.0a0
- SHARP 2.5
- APEX
- Nsight Compute 2021.3.0.13
- Nsight Systems 2021.3.2.4
- TensorBoard 2.7.0
- DALI 1.8
- MAGMA 2.5.2
- DLProf 1.8.0

- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 2.3.2
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 21.12 is based on NVIDIA CUDA 11.5.0, which requires NVIDIA Driver release 495 or later. However, if you are running on a Data Center GPU (for example, T4 or any other Tesla board), you may use NVIDIA driver release 418.40 (or later R418), 440.33 (or later R440), 450.51 (or later R450), 460.27 (or later R460), or 470.57 (or later R470). The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades and NVIDIA CUDA and Drivers Support.

GPU Requirements

Release 21.12 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, Turing, and NVIDIA Ampere GPU architecture families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 21.12 is based on 1.11.0a0+b6df043.
- The 21.12 container ships with a preview of the cuDNN v8 API and can be enabled via `` `export CUDNN_V8_API_ENABLED=1` ``. To use the new neural network-based heuristics, use `` `export USE_HEURISTIC_MODE_B=1` `` in addition to `` `export CUDNN_V8_API_ENABLED=1` ``. Please refer to the cuDNN API docs for more information about this heuristic mode (https://docs.nvidia.com/deeplearning/cudnn/api/index.html).

Announcements

- DLProf v1.8, which is included in the 21.12 container, will be the last release of DLProf. Starting with the 22.01 container, DLProf will no longer be included. It can still be manually installed via a pip wheel on the nvidia-pyindex.

- A preview of Torch-TensorRT (1.1.0a0) is now included. Torch-TRT is the TensorRT integration for PyTorch bringing the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.
- Starting with the 21.10 release, a beta version of the PyTorch container is available for the ARM SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container. To profile models in PyTorch, please use NVIDIA Deep Learning Profiler (DLProf). DLProf can help data scientists, engineers and researchers understand and improve performance of their models with visualization via DLProf Viewer in the web browser, or by analyzing text reports. DL Prof is available on NGC or a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`). Instead they can be obtained from Github or the NVIDIA GPU Cloud (NGC). Some python packages, included in previous containers to support these example models, have also been removed. Depending on their specific use cases, users may need to add some packages that were previously pre-installed.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.12	20.04	NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.2.1.8
21.11				TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for ARM SBSA Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub

- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA’s Mask R-CNN model is an optimized version of Facebook’s implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- The version of OpenUCX included with PyTorch container image version 21.11 has known issues with RAPIDS UCX-Py. When using Dask with this container version, pass `protocol="tcp"` to `LocalCUDACluster()`, not `protocol="ucx"`, to work around these issues. Additionally, LocalCUDACluster UCX-specific configurations must remain unspecified; they are: `enable_tcp_over_ucx`, `enable_nvlink`, `enable_infiniband`, `enable_rdmacm` and `ucx_net_devices`.
- ARM
  - Passing external CUDA Streams to PyTorch via ``` `torch.cuda.streams.ExternalStream(stream_v)` ``` might fail and is being debugged.

Chapter 63. PyTorch Release 21.11

The NVIDIA container image for PyTorch, release 21.11, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8 environment
- NVIDIA CUDA 11.5.0
- cuBLAS 11.7.3.1
- NVIDIA cuDNN 8.3.0.96
- NVIDIA NCCL 2.11.4 (optimized for NVLink™)
- RAPIDS 21.08
- rdma-core 36.0
- OpenMPI 4.1.2a1
- OpenUCX 1.11.0rc1
- GDRCopy 2.3
- NVIDIA HPC-X 2.9
- TensorRT 8.0.3.4 for x64 Linux
- TensorRT 8.0.2.2 for ARM SBSA Linux
- Torch-TensorRT 1.0.0a0
- SHARP 2.5
- APEX
- Nsight Compute 2021.3.0.13
- Nsight Systems 2021.3.2.4
- TensorBoard 2.7.0
- DALI 1.7
- MAGMA 2.5.2

- DLProf 1.7.0
- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 2.3.2
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 21.11 is based on NVIDIA CUDA 11.5.0, which requires NVIDIA Driver release 495 or later. However, if you are running on a Data Center GPU (for example, T4 or any other Tesla board), you may use NVIDIA driver release 418.40 (or later R418), 440.33 (or later R440), 450.51 (or later R450), 460.27 (or later R460), or 470.57 (or later R470). The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades and NVIDIA CUDA and Drivers Support.

GPU Requirements

Release 21.11 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, Turing, and NVIDIA Ampere GPU architecture families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 21.11 is based on 1.11.0a0+b6df043.
- Preview of Torch-TensorRT (1.0.0a0): The TensorRT integration for PyTorch.
- PyTorch container image version 21.11 introduces RAPIDS libraries cuDF, cuML, cuGraph, RMM, and XGBoost.

Announcements

- DLProf v1.8, which will be included in the 21.12 container, will be the last release of DLProf. Starting with the 22.01 container, DLProf will no longer be included. It can still be manually installed via a pip wheel on the nvidia-pyindex.
- A preview of Torch-TensorRT (1.0.0a0) is now included. Torch-TRT is the TensorRT integration for PyTorch bringing the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.

- Starting with the 21.10 release, a beta version of the PyTorch container is available for the ARM SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container. To profile models in PyTorch, please use NVIDIA Deep Learning Profiler (DLProf). DLProf can help data scientists, engineers and researchers understand and improve performance of their models with visualization via DLProf Viewer in the web browser, or by analyzing text reports. DL Prof is available on NGC or a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`). Instead they can be obtained from Github or the NVIDIA GPU Cloud (NGC). Some python packages, included in previous containers to support these example models, have also been removed. Depending on their specific use cases, users may need to add some packages that were previously pre-installed.
- A preview of Torch-TensorRT (1.0.0a0) is now included. Torch-TRT is the TensorRT integration for PyTorch bringing the capabilities of TensorRT directly to Torch in one line Python and C++ APIs.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.11	20.04	NVIDIA CUDA 11.5.0	1.11.0a0+b6df043	TensorRT 8.0.3.4 for x64 Linux
21.10		NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	TensorRT 8.0.2.2 for ARM SBSA Linux
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub

- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- The version of OpenUCX included with PyTorch container image version 21.11 has known issues with RAPIDS UCX-Py. When using Dask with this container version, pass `protocol="tcp"` to `LocalCUDACluster()`, not `protocol="ucx"`, to work around these issues. Additionally, LocalCUDACluster UCX-specific configurations must remain unspecified; they are: `enable_tcp_over_ucx`, `enable_nvlink`, `enable_infiniband`, `enable_rdmacm` and `ucx_net_devices`.
- ARM
  - Passing external CUDA Streams to PyTorch via `` `torch.cuda.streams.ExternalStream(stream_v)` `` might fail and is being debugged.

Chapter 64. PyTorch Release 21.10

The NVIDIA container image for PyTorch, release 21.10, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8 environment
- NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2
- NVIDIA cuDNN 8.2.4.15
- NVIDIA NCCL 2.11.4 (optimized for NVLink™)
- rdma-core 36.0
- OpenMPI 4.1.2a1
- OpenUCX 1.11.0rc1
- GDRCopy 2.3
- NVIDIA HPC-X 2.9
- Nsight Systems 2021.3.2.4
- TensorRT 8.0.3.4 for x64 Linux
- TensorRT 8.0.2.2 for ARM SBSA Linux
- SHARP 2.5
- APEX
- Nsight Compute 2021.2.2.0
- Nsight Systems 2021.3.2.4
- TensorBoard 2.6.0
- DALI 1.6
- MAGMA 2.5.2
- DLProf 1.6.0
- Jupyter and JupyterLab:

- Jupyter Client 6.0.0
- Jupyter Core 4.6.1
- Jupyter Notebook 6.0.3
- JupyterLab 2.3.2
- JupyterLab Server 1.0.6
- Jupyter-TensorBoard

Driver Requirements

Release 21.10 is based on NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2, which requires NVIDIA Driver release 470 or later. However, if you are running on Data Center GPUs (formerly Tesla), for example, T4, you may use NVIDIA driver release 418.40 (or later R418), 440.33 (or later R440), 450.51 (or later R450), or 460.27 (or later R460). The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades and NVIDIA CUDA and Drivers Support.

GPU Requirements

Release 21.10 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, Turing, and NVIDIA Ampere GPU architecture families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 21.10 is based on 1.10.0a0+0aef44c

Announcements

- Starting with the 21.10 release, a beta version of the PyTorch container is available for the ARM SBSA platform.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container. To profile models in PyTorch, please use NVIDIA Deep Learning Profiler (DLProf). DLProf can help data scientists, engineers and researchers understand and improve performance of their models with visualization via DLProf Viewer in the web browser, or by analyzing text reports. DL Prof is available on NGC or a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`). Instead they can be obtained

from Github or the NVIDIA GPU Cloud (NGC). Some python packages, included in previous containers to support these example models, have also been removed. Depending on their specific use cases, users may need to add some packages that were previously pre-installed.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.10	20.04	NVIDIA CUDA 11.4.2 with cuBLAS 11.6.5.2	1.10.0a0+0aef44c	TensorRT 8.0.3.4 for x64 Linux TensorRT 8.0.2.2 for ARM SBSA Linux
21.09		NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub
- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation,

leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- ARM
  - Passing external CUDA Streams to PyTorch via `` `torch.cuda.streams.ExternalStream(stream_v)` `` might fail and is being debugged.

Chapter 65. PyTorch Release 21.09

The NVIDIA container image for PyTorch, release 21.09, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8 environment
- NVIDIA CUDA 11.4.2
- cuBLAS 11.6.1.51
- NVIDIA cuDNN 8.2.4.15
- NVIDIA NCCL 2.11.4
- APEX
- rdma-core 36.0
- OpenMPI 4.1.2a1
- OpenUCX 1.11.0rc
- GDRCopy 2.3
- NVIDIA HPC-X 2.9
- Nsight Compute 2021.2.2.0
- Nsight Systems 2021.3.1.57
- TensorRT 8.0.3
- TensorBoard 2.6.0
- DALI 1.5
- MAGMA 2.5.2
- DLProf 1.5.0
- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1

- Jupyter Notebook 6.0.3
- JupyterLab 2.3.1
- JupyterLab Server 1.0.6
- Jupyter-TensorBoard

Driver Requirements

Release 21.09 is based on NVIDIA CUDA 11.4.2, which requires NVIDIA Driver release 470 or later. However, if you are running on Data Center GPUs (formerly Tesla), for example, T4, you may use NVIDIA driver release 418.40 (or later R418), 440.33 (or later R440), 450.51 (or later R450), or 460.27 (or later R460). The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades and NVIDIA CUDA and Drivers Support.

GPU Requirements

Release 21.09 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, Turing, and NVIDIA Ampere GPU architecture families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 21.09 is based on 1.10.0a0+3fd9dcf

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container. To profile models in PyTorch, please use NVIDIA Deep Learning Profiler (DLProf). DLProf can help data scientists, engineers and researchers understand and improve performance of their models with visualization via DLProf Viewer in the web browser, or by analyzing text reports. DL Prof is available on NGC or a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`). Instead they can be obtained from Github or the NVIDIA GPU Cloud (NGC). Some python packages, included in previous containers to support these example models, have also been removed. Depending on their specific use cases, users may need to add some packages that were previously pre-installed

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.09	20.04	NVIDIA CUDA 11.4.2	1.10.0a0+3fd9dcf	TensorRT 8.0.3
21.08		NVIDIA CUDA 11.4.1		TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12			1.4.0a0+174e1ba	TensorRT 6.0.1

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
19.11				
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub
- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- Known performance regressions in 21.09 vs. 21.07:
  - Up to 20% performance drop for Tacotron inference due to missing fused kernels in the scripted model.

Chapter 66. PyTorch Release 21.08

The NVIDIA container image for PyTorch, release 21.08, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8 environment
- NVIDIA CUDA 11.4.1
- cuBLAS 11.5.4
- NVIDIA cuDNN 8.2.2.26
- NVIDIA NCCL 2.10.3
- APEX
- rdma-core 36.0
- OpenMPI 4.1.2a1
- OpenUCX 1.11.0rc
- GDRCopy 2.2
- NVIDIA HPC-X 2.9
- Nsight Systems 2021.2.4.12
- TensorRT 8.0.1.6
- TensorBoard 2.6.0
- DALI 1.4
- MAGMA 2.5.2
- DLProf 1.4.0
- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3

- JupyterLab 2.3.1
- JupyterLab Server 1.0.6
- Jupyter-TensorBoard

Driver Requirements

Release 21.08 is based on NVIDIA CUDA 11.4.1, which requires NVIDIA Driver release 470 or later. However, if you are running on Data Center GPUs (formerly Tesla), for example, T4, you may use NVIDIA driver release 418.40 (or later R418), 440.33 (or later R440), 450.51 (or later R450), or 460.27 (or later R460). The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades and NVIDIA CUDA and Drivers Support.

GPU Requirements

Release 21.08 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, Turing, and NVIDIA Ampere GPU architecture families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 21.08 is based on 1.10.0a0+3fd9dcf

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container. To profile models in PyTorch, please use NVIDIA Deep Learning Profiler (DLProf). DLProf can help data scientists, engineers and researchers understand and improve performance of their models with visualization via DLProf Viewer in the web browser, or by analyzing text reports. DL Prof is available on NGC or a Python PIP wheel installation.
- The TensorCore example models are no longer provided in the core PyTorch container (previously shipped in `/workspace/nvidia-examples`). Instead they can be obtained from Github or the NVIDIA GPU Cloud (NGC). Some python packages, included in previous containers to support these example models, have also been removed. Depending on their specific use cases, users may need to add some packages that were previously pre-installed

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.08	20.04	NVIDIA CUDA 11.4.1	1.10.0a0+3fd9dcf	TensorRT 8.0.1.6
21.07		NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12			1.4.0a0+174e1ba	TensorRT 6.0.1
19.11				

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on

regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.

- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub
- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and

is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- Known performance regressions in 21.08 vs. 21.07:
  - Up to 20% performance drop for Tacotron inference due to missing fused kernels in the scripted model.

Chapter 67. PyTorch Release 21.07

The NVIDIA container image for PyTorch, release 21.07, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8 environment
- NVIDIA CUDA 11.4.0
- cuBLAS 11.5.2.43
- NVIDIA cuDNN 8.2.2.26
- NVIDIA NCCL 2.10.3
- APEX
- rdma-core 32.1
- OpenMPI 4.1.1rc1
- OpenUCX 1.10.1
- GDRCopy 2.2
- NVIDIA HPC-X 2.8.2rc3
- Nsight Compute 2021.2.0.0
- Nsight Systems 2021.2.4.12
- TensorRT 8.0.1.6
- TensorBoard 2.5.0
- DALI 1.3
- MAGMA 2.5.2
- DLProf 1.3.0
- Tensor Core optimized examples:
  - ResNeXt101-32x4d
  - SE-ResNext

- TransformerXL
- Jasper
- BERT
- Mask R-CNN
- Tacotron 2 and WaveGlow v1.1
- SSD300 v1.1
- Neural Collaborative Filtering (NCF)
- ResNet50 v1.5
- GNMT v2

- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 2.3.1
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 21.07 is based on NVIDIA CUDA 11.4.0, which requires NVIDIA Driver release 470 or later. However, if you are running on Data Center GPUs (formerly Tesla), for example, T4, you may use NVIDIA driver release 418.40 (or later R418), 440.33 (or later R440), 450.51 (or later R450), or 460.27 (or later R460). The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades and NVIDIA CUDA and Drivers Support.

GPU Requirements

Release 21.07 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, Turing, and NVIDIA Ampere GPU architecture families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 21.07 is based on 1.10.0a0+ecc3718

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container. To profile models in PyTorch, please use NVIDIA Deep Learning Profiler (DLProf). DLProf can help data scientists, engineers and researchers understand and improve performance of their models with visualization via DLProf Viewer in the web browser, or by analyzing text reports. DL Prof is available on NGC or a Python PIP wheel installation.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.07	20.04	NVIDIA CUDA 11.4.0	1.10.0a0+ecc3718	TensorRT 8.0.1.6
21.06		NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub
- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions

paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- The 21.07 release includes libsystemd and libudev versions that have a known vulnerability that was discovered late in our QA process. See CVE-2021-33910 for details. This will be fixed in the next release.

Chapter 68. PyTorch Release 21.06

The NVIDIA container image for PyTorch, release 21.06, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8 environment
- NVIDIA CUDA 11.3.1
- cuBLAS 11.5.1.109
- NVIDIA cuDNN 8.2.1
- NVIDIA NCCL 2.9.9 (optimized for NVLink™)
- APEX
- rdma-core 32.1
- OpenMPI 4.1.1rc1
- OpenUCX 1.10.1
- GDRCopy 2.2
- NVIDIA HPC-X 2.8.2rc3
- Nsight Compute 2021.1.0.0
- Nsight Systems 2021.2.1.58
- TensorRT 7.2.3.4
- TensorBoard 1.15.5
- DALI 1.2
- MAGMA 2.5.2
- DLProf 1.2.0
- PyProf r21.06
- Tensor Core optimized examples:
  - ResNeXt101-32x4d

- SE-ResNext
- TransformerXL
- Jasper
- BERT
- Mask R-CNN
- Tacotron 2 and WaveGlow v1.1
- SSD300 v1.1
- Neural Collaborative Filtering (NCF)
- ResNet50 v1.5
- GNMT v2

- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 2.3.1
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 21.06 is based on NVIDIA CUDA 11.3.1, which requires NVIDIA Driver release 465.19.01 or later. However, if you are running on Data Center GPUs (formerly Tesla), for example, T4, you may use NVIDIA driver release 418.40 (or later R418), 440.33 (or later R440), 450.51 (or later R450), or 460.27 (or later R460). The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades and NVIDIA CUDA and Drivers Support.

GPU Requirements

Release 21.06 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, Turing, and NVIDIA Ampere GPU architecture families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 21.06 is based on 1.9.0a0+c3d40fd
- Ubuntu 20.04 with May 2021 updates

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container. To profile models in PyTorch, please use NVIDIA Deep Learning Profiler (DLProf). DLProf can help data scientists, engineers and researchers understand and improve performance of their models with visualization via DLProf Viewer in the web browser, or by analyzing text reports. DL Prof is available on NGC or a Python PIP wheel installation.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.06	20.04	NVIDIA CUDA 11.3.1	1.9.0a0+c3d40fd	TensorRT 7.2.3.4
21.05		NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model

is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub
- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based

Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- Known performance regressions in 21.06 vs. 21.05:
  - On Turing and NVIDIA Ampere Architecture GPUs:
    - Up to 15% performance drop for GNMT training
  - On Volta:
    - Up to 20% performance drop for Tacotron training.
- Manual synchronization is required in CUDA graphs workloads between graph replays.
- The PyTorch container includes a version of Django with a known vulnerability that was discovered late in our QA process. See CVE-2021-31542 for details. This will be fixed in the next release.
- The PyTorch container includes a version of Pillow with known vulnerabilities discovered late in our QA process. See CVE-2021-25287, CVE-2021-28676, CVE-2021-28677, and CVE-2021-25288 for details. This will be fixed in the next release.

Chapter 69. PyTorch Release 21.05

The NVIDIA container image for PyTorch, release 21.05, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8 environment
- NVIDIA CUDA 11.3.0
- cuBLAS 11.5.1.101
- NVIDIA cuDNN 8.2.0.51
- NVIDIA NCCL 2.9.8 (optimized for NVLink™)
- APEX
- rdma-core 32.1
- OpenMPI 4.1.1rc1
- OpenUCX 1.10.0
- GDRCopy 2.2
- NVIDIA HPC-X 2.8.2rc3
- Nsight Compute 2021.1.0.0
- Nsight Systems 2021.1.3.14
- TensorRT 7.2.3.4
- TensorBoard 1.15.5
- DALI 1.0.0
- MAGMA 2.5.2
- DLProf 1.1.0
- PyProf r21.05
- Tensor Core optimized examples:
  - ResNeXt101-32x4d

- SE-ResNext
- TransformerXL
- Jasper
- BERT
- Mask R-CNN
- Tacotron 2 and WaveGlow v1.1
- SSD300 v1.1
- Neural Collaborative Filtering (NCF)
- ResNet50 v1.5
- GNMT v2

- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 2.3.1
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 21.05 is based on NVIDIA CUDA 11.3.0, which requires NVIDIA Driver release 465.19.01 or later. However, if you are running on Data Center GPUs (formerly Tesla), for example, T4, you may use NVIDIA driver release 418.40 (or later R418), 440.33 (or later R440), 450.51 (or later R450), or 460.27 (or later R460). The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades and NVIDIA CUDA and Drivers Support.

GPU Requirements

Release 21.05 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, Turing, and NVIDIA Ampere GPU architecture families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 21.05 is based on 1.9.0a0+2ecb2c7
- Ubuntu 20.04 with April 2021 updates

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Starting in 21.06, PyProf will no longer be included in the NVIDIA PyTorch container. To profile models in PyTorch, please use NVIDIA Deep Learning Profiler (DLProf). DLProf can help data scientists, engineers and researchers understand and improve performance of their models with visualization via DLProf Viewer in the web browser, or by analyzing text reports. DL Prof is available on NGC or a Python PIP wheel installation.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.05	20.04	NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	TensorRT 7.2.3.4
21.04				
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12			1.4.0a0+174e1ba	TensorRT 6.0.1

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
19.11				
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub
- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- Known performance regressions in 21.05 vs. 21.04:
  - On NVIDIA Ampere Architecture GPUs:
    - Up to 17% performance drop for VGG16 training
- Manual synchronization is required in CUDA graphs workloads between graph replays.
- The DLProf TensorBoard plugin included with the 21.04 and 21.05 releases is an incorrect version with respect to the DLProf command line tool included in those releases. To correct this, use the following command:

```
$ pip install --index-urlhttps://developer.download.nvidia.com/compute/redist
 nvidia_tensorboard_plugin_dlprof==1.1.0
```

Chapter 70. PyTorch Release 21.04

The NVIDIA container image for PyTorch, release 21.04, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8 environment
- NVIDIA CUDA 11.3.0
- cuBLAS 11.5.1.101
- NVIDIA cuDNN 8.2.0.41
- NVIDIA NCCL 2.9.6 (optimized for NVLink™)
- APEX
- rdma-core 32.1
- OpenMPI 4.1.1rc1
- OpenUCX 1.10.0
- GDRCopy 2.2
- NVIDIA HPC-X 2.8.2rc3
- Nsight Compute 2021.1.0.18
- Nsight Systems 2021.1.3.14
- TensorRT 7.2.3.4
- TensorBoard 1.15.5
- DALI 1.0.0
- MAGMA 2.5.2
- DLProf 1.1.0
- PyProf r21.04
- Tensor Core optimized examples:
  - ResNeXt101-32x4d

- SE-ResNext
- TransformerXL
- Jasper
- BERT
- Mask R-CNN
- Tacotron 2 and WaveGlow v1.1
- SSD300 v1.1
- Neural Collaborative Filtering (NCF)
- ResNet50 v1.5
- GNMT v2

- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 2.3.1
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 21.04 is based on NVIDIA CUDA 11.3.0, which requires NVIDIA Driver release 465.19.01 or later. However, if you are running on Data Center GPUs (formerly Tesla), for example, T4, you may use NVIDIA driver release 418.40 (or later R418), 440.33 (or later R440), 450.51 (or later R450), or 460.27 (or later R460). The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades and NVIDIA CUDA and Drivers Support.

GPU Requirements

Release 21.04 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, Turing, and NVIDIA Ampere GPU architecture families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 21.04 is based on 1.9.0a0+2ecb2c7
- Experimental release of the nvfuser backend for scripted models. Users can enable it using the context manager: `with.torch.jit.fuser("fuser2"):`

- Ubuntu 20.04 with March 2021 updates

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Transformer has been removed.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.04	20.04	NVIDIA CUDA 11.3.0	1.9.0a0+2ecb2c7	TensorRT 7.2.3.4
21.03		NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.

- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub
- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA’s Mask R-CNN model is an optimized version of Facebook’s implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- Known performance regressions in 21.04 vs. 21.03:
  - On NVIDIA Ampere Architecture GPUs:
    - Up to 38% performance drop for FastPitch training
    - Up to 30% performance drop in MaskRCNN training
  - On Turing:
    - Up to 20% performance drop in MaskRCNN training
    - Up to 15% performance drop in VGG16 training
- Manual synchronization is required in CUDA graphs workloads between graph replays.
- The DLProf TensorBoard plugin included with the 21.04 release is an incorrect version with respect to the DLProf command line tool included in those releases. To correct this, use the following command:

```
$ pip install --index-urlhttps://developer.download.nvidia.com/compute/redist
 nvidia_tensorboard_plugin_dlprof==1.1.0
```

Chapter 71. PyTorch Release 21.03

The NVIDIA container image for PyTorch, release 21.03, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8 environment
- NVIDIA CUDA 11.2.1 including cuBLAS 11.4.1.1026
- NVIDIA cuDNN 8.1.1
- NVIDIA NCCL 2.8.4 (optimized for NVLink™)
- APEX
- MLNX_OFED
- OpenMPI 4.0.5
- TensorBoard 1.15.5
- Nsight Compute 2020.3.1.0
- Nsight Systems 2020.4.3.7
- TensorRT 7.2.2.3
- DALI 0.31.0
- MAGMA 2.5.2
- DLProf 1.0.0
- PyProf r21.03
- Tensor Core optimized examples:
  - ResNeXt101-32x4d
  - SE-ResNext
  - TransformerXL
  - Jasper
  - BERT

- Mask R-CNN
- Tacotron 2 and WaveGlow v1.1
- SSD300 v1.1
- Neural Collaborative Filtering (NCF)
- ResNet50 v1.5
- GNMT v2

- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 1.2.0
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 21.03 is based on NVIDIA CUDA 11.2.1, which requires NVIDIA Driver release 460.32.03 or later. However, if you are running on Data Center GPUs (formerly Tesla), for example, T4, you may use NVIDIA driver release 418.40 (or later R418), 440.33 (or later R440), 450.51(or later R450). The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades and NVIDIA CUDA and Drivers Support.

GPU Requirements

Release 21.03 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, Turing, and NVIDIA Ampere GPU architecture families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 21.03 is based on 1.9.0a0+df837d0
- NVIDIA CUDA 11.2.1 including cuBLAS 11.4.1.1026
- The latest version of NVIDIA cuDNN 8.1.1
- The latest version of DALI 0.31.0
- The latest version of DLProf 1.0.0
- The latest version of PyProf r21.03
- Ubuntu 20.04 with February 2021 updates

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Transformer has been removed.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.03	20.04	NVIDIA CUDA 11.2.1	1.9.0a0+df837d0	TensorRT 7.2.2.3
21.02		NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up	TensorRT 5.1.5

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
			through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.

- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub
- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user

interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

Known performance regressions in 21.03 vs. 21.02:

- On A102 (e.g. A40):
  - Up to 28% performance drop for BERT large inference and pre-training
  - Up to 33% performance drop for Transformer training
- On TU102 (e.g. RTX6000):
  - Up to 18% performance drop for inference on ResNet-like models
  - Up to 25% performance drop for SSD inference
  - Up to 15% performance drop for WaveGlow

Chapter 72. PyTorch Release 21.02

The NVIDIA container image for PyTorch, release 21.02, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8 environment
- NVIDIA CUDA 11.2.0 including cuBLAS 11.3.1
- NVIDIA cuDNN 8.1.0
- NVIDIA NCCL 2.8.4 (optimized for NVLink™)
- APEX
- MLNX_OFED
- OpenMPI 4.0.5
- TensorBoard 1.15.5
- Nsight Compute 2020.3.0.18
- Nsight Systems 2020.4.3.7
- TensorRT 7.2.2.3+cuda11.1.0.024
- DALI 0.29.0
- MAGMA 2.5.2
- DLProf 0.19.0
- PyProf r21.02
- Tensor Core optimized examples:
  - ResNeXt101-32x4d
  - SE-ResNext
  - TransformerXL
  - Jasper
  - BERT

- Mask R-CNN
- Tacotron 2 and WaveGlow v1.1
- SSD300 v1.1
- Neural Collaborative Filtering (NCF)
- ResNet50 v1.5
- GNMT v2

- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 1.2.0
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 21.02 is based on NVIDIA CUDA 11.2.0, which requires NVIDIA Driver release 460.27.04 or later. However, if you are running on Data Center GPUs (formerly Tesla), for example, T4, you may use NVIDIA driver release 418.40 (or later R418), 440.33 (or later R440), 450.51(or later R450). The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades and NVIDIA CUDA and Drivers Support.

GPU Requirements

Release 21.02 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, Turing, and NVIDIA Ampere GPU architecture families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 21.02 is based on 1.8.0a0+52ea372
- NVIDIA CUDA 11.2.0 including cuBLAS 11.3.1
- The latest version of NVIDIA cuDNN 8.0.5
- The latest version of NVIDIA NCCL 2.8.4
- The latest version of Nsight Compute 2020.3.0.18
- The latest version of Nsight Systems 2020.4.3.7
- The latest version of TensorRT 7.2.2.3+cuda11.1.0.024

- The latest version of DALI 0.29
- The latest version of DLProf 0.19.0
- The latest version of PyProf r21.02
- Ubuntu 20.04 with January 2021 updates

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Transformer has been removed.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
21.02	20.04	NVIDIA CUDA 11.2.0	1.8.0a0+52ea372	TensorRT 7.2.2.3+cuda11.1.0.024
20.12		NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.

- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub
- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- Known performance regressions in 21.02 vs. 20.12 for:
  - SSD-Resnet-50 training on Ampere up to 8%
  - ResNext101 inference on Volta and Ampere up to 24%
  - Se-ResNeXt101 inference on Ampere up to 17%
  - SSD inference on Volta up to 17%

Chapter 73. PyTorch Release 21.01

The NVIDIA container image release for PyTorch 21.01 has been canceled. The next release will be the 21.02 release which is expected to be released at the end of February.

Chapter 74. PyTorch Release 20.12

The NVIDIA container image for PyTorch, release 20.12, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.8/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 20.04 including Python 3.8 environment
- NVIDIA CUDA 11.1.1 including cuBLAS 11.3.0
- NVIDIA cuDNN 8.0.5
- NVIDIA NCCL 2.8.3 (optimized for NVLink™)
- APEX
- MLNX_OFED
- OpenMPI 4.0.5
- TensorBoard 1.15.0+nv20.11
- Nsight Compute 2020.2.1.8
- Nsight Systems 2020.3.4.32
- TensorRT 7.2.2
- DALI 0.28.0
- MAGMA 2.5.2
- DLProf 0.18.0
- PyProf r20.12
- Tensor Core optimized examples:
  - ResNeXt101-32x4d
  - SE-ResNext
  - TransformerXL
  - Jasper
  - BERT

- Mask R-CNN
- Tacotron 2 and WaveGlow v1.1
- SSD300 v1.1
- Neural Collaborative Filtering (NCF)
- ResNet50 v1.5
- GNMT v2

- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 1.2.0
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 20.12 is based on NVIDIA CUDA 11.1.1, which requires NVIDIA Driver release 455 or later. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 418.xx, 440.30, or 450.xx. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 20.12 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, Turing, and NVIDIA Ampere GPU architecture families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 20.12 is based on 1.8.0a0+1606899
- NVIDIA CUDA 11.1.1 including cuBLAS 11.3.0
- The latest version of NVIDIA cuDNN 8.0.5
- The latest version of NVIDIA NCCL 2.8.3
- The latest version of Nsight Compute 2020.2.1.8
- The latest version of TensorRT 7.2.2
- The latest version of DALI 0.28

- The latest version of DLProf 0.18.0
- The latest version of PyProf r20.12
- Ubuntu 20.04 with November 2020 updates

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Transformer has been removed.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.12	20.04	NVIDIA CUDA 11.1.1	1.8.0a0+1606899	TensorRT 7.2.2
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit	TensorRT 5.1.5

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
			9130ab38 from July 31, 2019 as well as a cherry-picked	

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements

introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub

- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative

Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- A workaround for the WaveGlow training regression from our past containers is to use a fake batch dimension when calculating the log determinant via `torch.logdet(W.unsqueeze(0).float()).squeeze()` as is done in this release.
- Known performance regressions in 20.12 vs. 20.11 for:
  - MaskR-CNN training up to 15%
  - Transformer-XL inference of approx. 10%
  - Tacotron2+Waveglow inference up to 50%
  - FastPitch inference and training up to 15%

Chapter 75. PyTorch Release 20.11

The NVIDIA container image for PyTorch, release 20.11, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 18.04 including Python 3.6 environment
- NVIDIA CUDA 11.1.0 including cuBLAS 11.2.1
- NVIDIA cuDNN 8.0.4
- NVIDIA NCCL 2.8.2 (optimized for NVLink™)
- APEX
- MLNX_OFED 5.1
- OpenMPI 4.0.5
- TensorBoard 1.15.0+nv20.11
- Nsight Compute 2020.2.0.18
- Nsight Systems 2020.3.4.32
- TensorRT 7.2.1
- DALI 0.27.0
- MAGMA 2.5.2
- DLProf 0.17.0
- PyProf r20.11
- Tensor Core optimized examples:
  - ResNeXt101-32x4d
  - SE-ResNext
  - TransformerXL
  - Jasper
  - BERT

- Mask R-CNN
- Tacotron 2 and WaveGlow v1.1
- SSD300 v1.1
- Neural Collaborative Filtering (NCF)
- ResNet50 v1.5
- GNMT v2

- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 1.2.0
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 20.11 is based on NVIDIA CUDA 11.1.0, which requires NVIDIA Driver release 455 or later. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 418.xx, 440.30, or 450.xx. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 20.11 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, Turing, and NVIDIA Ampere GPU architecture families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 20.11 is based on 1.8.0a0+17f8c32
- The latest version of NVIDIA NCCL 2.8.2
- The latest version of DALI 0.27
- The latest version of DLProf 0.17.0
- The latest version of PyProf r20.11
- Ubuntu 18.04 with October 2020 updates

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Transformer has been removed.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.11	18.04	NVIDIA CUDA 11.1.0	1.8.0a0+17f8c32	TensorRT 7.2.1
20.10			1.7.0a0+7036e91	
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub

- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA’s Mask R-CNN model is an optimized version of Facebook’s implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- The PyProf version string in the 20.11 container is truncated to `3.` and does not display the full version string as `3.6.0`.
- A workaround for the WaveGlow training regression from our past containers is to use a fake batch dimension when calculating the log determinant via `torch.logdet(W.unsqueeze(0).float()).squeeze()` as is done in this release.
- Known performance regressions in 20.11 vs. 20.09 for:
  - MaskR-CNN training up to 9%
  - Transformer mixed-precision training up to 11%
  - Transformer-XL training and inference of approx. 10%
  - ResNet and ResNext inference up to 17%
- Known performance regressions in 20.11 vs. 2010 for:
  - GNMT mixed-precision training and inference up to 20%

Chapter 76. PyTorch Release 20.10

The NVIDIA container image for PyTorch, release 20.10, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 18.04 including Python 3.6 environment
- NVIDIA CUDA 11.1.0 including cuBLAS 11.2.1
- NVIDIA cuDNN 8.0.4
- NVIDIA NCCL 2.7.8 (optimized for NVLink™)
- APEX
- MLNX_OFED
- OpenMPI 4.0.5
- TensorBoard 1.15.0+nv
- Nsight Compute 2020.2.0.18
- Nsight Systems 2020.3.4.32
- TensorRT 7.2.1
- DALI 0.26.0
- MAGMA 2.5.2
- DLProf 0.16.0
- PyProf r20.10
- Tensor Core optimized examples:
  - ResNeXt101-32x4d
  - SE-ResNext
  - TransformerXL
  - Jasper
  - BERT

- Mask R-CNN
- Tacotron 2 and WaveGlow v1.1
- SSD300 v1.1
- Neural Collaborative Filtering (NCF)
- ResNet50 v1.5
- GNMT v2

- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 1.2.0
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 20.10 is based on NVIDIA CUDA 11.1.0, which requires NVIDIA Driver release 455 or later. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 418.xx, 440.30, or 450.xx. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 20.10 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, Turing, and NVIDIA Ampere GPU architecture families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 20.10 is based on 1.7.0a0+7036e91
- The latest version of NVIDIA CUDA 11.1.0 including cuBLAS 11.2.1
- The latest version of NVIDIA cuDNN 8.0.4
- The latest version of TensorRT 7.2.1
- The latest version of OpenMPI 4.0.5
- The latest version of Nsight Compute 2020.2.0.18
- The latest version of Nsight Systems 2020.3.4.32

- The latest version of DALI 0.26
- The latest version of DLProf 0.16.0
- The latest version of PyProf 3.5.0
- Ubuntu 18.04 with September 2020 updates

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Transformer has been removed.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.10	18.04	NVIDIA CUDA 11.1.0	1.7.0a0+7036e91	TensorRT 7.2.1
20.09		NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from	TensorRT 5.1.5

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
			July 31, 2019 as well as a cherry-picked	

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by

attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub

- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative

Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- The 20.10 container ships with libfreetype6 2.8.1-2ubuntu2.1, which is vulnerable to CVE-2020-15999. Use

```
$ apt-get update
$ apt --only-upgrade install libfreetype6
```

  to apply the patch to fix this issue, or alternatively, purge this package from the container.
- A workaround for the WaveGlow training regression from our past containers is to use a fake batch dimension when calculating the log determinant via `torch.logdet(W.unsqueeze(0).float()).squeeze()` as is done in this release.
- Known performance regressions in 20.10 vs. 20.09 for:
  - Bert mixed-precision training up to 13%
  - MaskR-CNN training up to 9%
  - Transformer mixed-precision training up to 14%
  - Transformer-XL training and inference of approx. 10%
  - ResNet and ResNext inference up to 17%
  - FastPitch FP32 training up to 13%

Chapter 77. PyTorch Release 20.09

The NVIDIA container image for PyTorch, release 20.09, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 18.04 including Python 3.6 environment
- NVIDIA CUDA 11.0.3 including cuBLAS 11.2.0
- NVIDIA cuDNN 8.0.4
- NVIDIA NCCL 2.7.8 (optimized for NVLink™)
- APEX
- MLNX_OFED
- OpenMPI 3.1.6
- TensorBoard 1.15.0+nv
- Nsight Compute 2020.1.2.4
- Nsight Systems 2020.3.2.6
- TensorRT 7.1.3
- DALI 0.25.1
- MAGMA 2.5.2
- DLProf 0.15.0
- PyProf r20.09
- Tensor Core optimized examples:
  - ResNeXt101-32x4d
  - SE-ResNext
  - TransformerXL
  - Jasper
  - BERT

- Mask R-CNN
- Tacotron 2 and WaveGlow v1.1
- SSD300 v1.1
- Neural Collaborative Filtering (NCF)
- ResNet50 v1.5
- GNMT v2

- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 1.2.0
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 20.09 is based on NVIDIA CUDA 11.0.3, which requires NVIDIA Driver release 450 or later. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 418.xx or 440.30. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 20.09 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 20.09 is based on 1.7.0a0+8deb4fe with minor cherry-picked bug fixes
- The latest version of NVIDIA cuDNN 8.0.4
- The latest version of DALI 0.25.1
- The latest version of DLProf 0.15.0
- The latest version of PyProf r20.09
- Ubuntu 18.04 with August 2020 updates

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Transformer has been removed.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.09	18.04	NVIDIA CUDA 11.0.3	1.7.0a0+8deb4fe	TensorRT 7.1.3
20.08			1.7.0a0+6392713	
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try

mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub
- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among

end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- A workaround for the WaveGlow training regression from our past containers is to use a fake batch dimension when calculating the log determinant via `torch.logdet(W.unsqueeze(0).float()).squeeze()` as is done in this release.
- Known performance regressions in 20.08 vs. 20.07 for ResNet- and ResNext-like models on all architectures due to a temporal workaround in the dispatching mechanism (commit 0494e0a) up to 21%.
- Known NVIDIA Ampere GPU architecture performance regressions for FastPitch training in FP32 in 20.09 vs. 20.08 up to 10%.

Chapter 78. PyTorch Release 20.08

The NVIDIA container image for PyTorch, release 20.08, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 18.04 including Python 3.6 environment
- NVIDIA CUDA 11.0.3 including cuBLAS 11.2.0
- NVIDIA cuDNN 8.0.2
- NVIDIA NCCL 2.7.8 (optimized for NVLink™)
- APEX
- MLNX_OFED
- OpenMPI 3.1.6
- TensorBoard 1.15.0+nv
- Nsight Compute 2020.1.2.4
- Nsight Systems 2020.3.2.6
- TensorRT 7.1.3
- DALI 0.24
- MAGMA 2.5.2
- DLProf 0.14.0
- PyProf r20.08
- Tensor Core optimized examples:
  - ResNeXt101-32x4d
  - SE-ResNext
  - TransformerXL
  - Jasper
  - BERT

- Mask R-CNN
- Tacotron 2 and WaveGlow v1.1
- SSD300 v1.1
- Neural Collaborative Filtering (NCF)
- ResNet50 v1.5
- GNMT v2

- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 1.2.0
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 20.08 is based on NVIDIA CUDA 11.0.3, which requires NVIDIA Driver release 450 or later. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 418.xx or 440.30. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 20.08 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 20.08 is based on 1.7.0a0+6392713 with minor cherry-picked bug fixes
- The latest version of NVIDIA CUDA 11.0.3 including cuBLAS 11.2.0
- The latest version of NVIDIA NCCL 2.7.8
- The latest version of NVIDIA cuDNN 8.0.2
- The latest version of Nsight Compute 2020.1.2.4
- The latest version of Nsight Systems 2020.3.2.6
- The latest version of DALI 0.24

- The latest version of DLProf 0.14.0
- The latest version of PyProf r20.08
- Ubuntu 18.04 with July 2020 updates

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Transformer has been removed.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.08	18.04	NVIDIA CUDA 11.0.3	1.7.0a0+6392713	TensorRT 7.1.3
20.07		NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub

- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- A workaround for the WaveGlow training regression from our past containers is to use a fake batch dimension when calculating the log determinant via `torch.logdet(W.unsqueeze(0).float()).squeeze()` as is done in this release.
- Known performance regressions in 20.08 vs. 20.07 for ResNet- and ResNext-like models on all architectures due to a temporal workaround in the dispatching mechanism (commit 0494e0a) up to 18%.
- Known Turing performance regressions for FastPitch and WaveGlow inference in 20.08 vs. 20.07 up to 75%.

Chapter 79. PyTorch Release 20.07

The NVIDIA container image for PyTorch, release 20.07, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 18.04 including Python 3.6 environment
- NVIDIA CUDA 11.0.194 including cuBLAS 11.1.0
- NVIDIA cuDNN 8.0.1
- NVIDIA NCCL 2.7.6 (optimized for NVLink™)
- APEX
- MLNX_OFED
- OpenMPI 3.1.6
- TensorBoard 1.15.0+nv
- Nsight Compute 2020.1.1.8
- Nsight Systems 2020.3.2.6
- TensorRT 7.1.3
- DALI 0.23
- MAGMA 2.5.2
- DLProf 0.13.0
- PyProf r20.07
- Tensor Core optimized examples:
  - ResNeXt101-32x4d
  - SE-ResNext
  - TransformerXL
  - Jasper
  - BERT

- Mask R-CNN
- Tacotron 2 and WaveGlow v1.1
- SSD300 v1.1
- Neural Collaborative Filtering (NCF)
- ResNet50 v1.5
- GNMT v2

- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 1.2.0
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 20.07 is based on NVIDIA CUDA 11.0.194, which requires NVIDIA Driver release 450 or later. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 418.xx or 440.30. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 20.07 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 20.07 is based on 1.6.0a0+9907a3e with minor cherry-picked bug fixes
- The latest version of NVIDIA CUDA 11.0.194 including cuBLAS 11.1.0
- The latest version of NVIDIA NCCL 2.7.6
- The latest version of NVIDIA cuDNN 8.0.1
- The latest version of Nsight Compute 2020.1.1.8
- The latest version of Nsight Systems 2020.3.2.6
- The latest version of TensorRT 7.1.3

- The latest version of OpenMPI 3.1.6
- Ubuntu 18.04 with June 2020 updates
- Latest version of DALI 0.23

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Transformer has been removed.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.07	18.04	NVIDIA CUDA 11.0.194	1.6.0a0+9907a3e	TensorRT 7.1.3
20.06		NVIDIA CUDA 11.0.167		TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub

- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- There is up to 5% performance drop on Transformer-XL mixed precision training in the 20.07 container compared to 19.11. Disabling the profiling executor at the beginning of your script might reduce this effect via:

```
torch._C._jit_set_profiling_executor(False)
torch._C._jit_set_profiling_mode(False)
```

- A workaround for the WaveGlow training regression from our past containers is to use a fake batch dimension when calculating the log determinant via `torch.logdet(W.unsqueeze(0).float()).squeeze()` as is done in this release.
- Known Turing performance regressions in 20.07 vs. 20.03 container:
  - Up to 10% performance drop on InceptionV3 for mixed precision training
  - Up to 10% performance drop on MaskNCF FP32 training
  - Up to 25% performance drop on MaskRCNN FP32 training
  - Up to 10% performance drop on ResNet50 for mixed precision training.
- Known Volta performance regressions in 20.07 vs. 20.03 container:
  - Up to 30% performance drop on WaveGlow for FP32 training
  - Up to 11% performance drop on ResNet101 and ResNet152 mixed precision training
  - Up to 10% performance drop on full FP16 VGG16 training
- Known Pascal performance regressions in 20.07 vs. 20.03 container:
  - Up to 19% performance drop on MaskRCNN for FP32 training
- When FFT Tiled algo are used with 3D convolution, an intermittent silent failure might happen due to dependency on the order of the stream execution. In some cases this might be manifested as NaNs in the output and we recommend to disable cuDNN via `torch.backends.cudnn.enabled = False`.
- Channels-last memory format is experimental in the 20.07 container. Potential convergence issues for ResNet variants are being investigated. On NVIDIA Ampere architecture based GPUs unexpected NaN values due to a race condition in a cuDNN kernel might be observed. We recommend to use the default memory format in case you run into these issues.

Chapter 80. PyTorch Release 20.06

The NVIDIA container image for PyTorch, release 20.06, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 18.04 including Python 3.6 environment
- NVIDIA CUDA 11.0.167 including cuBLAS 11.1.0
- NVIDIA cuDNN 8.0.1
- NVIDIA NCCL 2.7.5 (optimized for NVLink™)
- APEX
- MLNX_OFED
- OpenMPI 3.1.6
- TensorBoard 1.15.0+nv
- Nsight Compute 2020.1.0.33
- Nsight Systems 2020.2.5.8
- TensorRT 7.1.2
- DALI 0.22
- MAGMA 2.5.2
- DLProf 0.12.0
- PyProf r20.06
- Tensor Core optimized examples:
  - ResNeXt101-32x4d
  - SE-ResNext
  - TransformerXL
  - Jasper
  - BERT

- Mask R-CNN
- Tacotron 2 and WaveGlow v1.1
- SSD300 v1.1
- Neural Collaborative Filtering (NCF)
- ResNet50 v1.5
- GNMT v2

- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 1.2.14
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 20.06 is based on NVIDIA CUDA 11.0.167, which requires NVIDIA Driver release 450 or later. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 418.xx or 440.30. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 20.06 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 20.06 is based on 1.6.0a0+9907a3e
- The latest version of NVIDIA CUDA 11.0.167 including cuBLAS 11.1.0
- The latest version of NVIDIA NCCL 2.7.5
- The latest version of NVIDIA cuDNN 8.0.1
- The latest version of Nsight Compute 2020.1.0.33
- The latest version of Nsight Systems 2020.2.5.8
- The latest version of TensorRT 7.1.2

- The latest version of OpenMPI 3.1.6
- Ubuntu 18.04 with May 2020 updates
- Latest version of DALI 0.22
- Native Automatic Mixed Precision (torch.cuda.amp). See official API documentation and examples. torch.cuda.amp is intended as the future-proof replacement for Apex AMP, and offers a number of advantages.
- Integrated latest NVIDIA Deep Learning SDK to support NVIDIA A100 using CUDA 11 and cuDNN 8
- Various bug fixes for channels-last layout optimization. Note that this layout is still in experimental form. See Known Issues below.
- Performance improvements for various torch.distribution methods by switching to the TensorIterator implementation
- Default TF32 support for Ampere-based GPUs

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Transformer has been removed.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.06	18.04	NVIDIA CUDA 11.0.167	1.6.0a0+9907a3e	TensorRT 7.1.2
20.03		NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up	TensorRT 5.1.5

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
			through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	

Automatic Mixed Precision (AMP)

Automatic Mixed Precision (AMP) for PyTorch is available in this container through the native implementation as well as a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Apex AMP is included to support models that currently rely on it, but torch.cuda.amp is the future-proof alternative, and offers a number of advantages over Apex AMP.

Guidance and examples demonstrating torch.cuda.amp can be found here.Apex AMP examples can be found here.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.

- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub
- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user

interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- There is up to 5% performance drop on Transformer-XL mixed precision training in the 20.01 container compared to 19.11. Disabling the profiling executor at the beginning of your script might reduce this effect via:

```
torch._C._jit_set_profiling_executor(False)
torch._C._jit_set_profiling_mode(False)
```

- A workaround for the WaveGlow training regression from our past containers is to use a fake batch dimension when calculating the log determinant via `torch.logdet(W.unsqueeze(0).float()).squeeze()` as is done in this release.
- Known Turing performance regressions in 20.06 vs. 20.03 container:
  - Up to 10% performance drop on InceptionV3 for mixed precision training
  - Up to 10% performance drop on MaskNCF FP32 training
  - Up to 25% performance drop on MaskRCNN FP32 training
  - Up to 10% performance drop on ResNet50 for mixed precision training.
- Known Volta performance regressions in 20.06 vs. 20.03 container:
  - Up to 30% performance drop on WaveGlow for FP32 training
  - Up to 11% performance drop on ResNet101 and ResNet152 mixed precision training
- Known Pascal performance regressions in 20.06 vs. 20.03 container:
  - Up to 19% performance drop on MaskRCNN for FP32 training
- When FFT Tiled algo are used with 3D convolution, an intermittent silent failure might happen due to dependency on the order of the stream execution. In some cases this might be manifested as NaNs in the output and we recommend to disable cuDNN via `torch.backends.cudnn.enabled = False`.
- Channels-last memory format is experimental in the 20.06 container. Potential convergence issues for ResNet variants are being investigated. On NVIDIA Ampere architecture based GPUs unexpected NaN values due to a race condition in a cuDNN

kernel might be observed. We recommend to use the default memory format in case you run into these issues.

Chapter 81. PyTorch Release 20.03

The NVIDIA container image for PyTorch, release 20.03, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 18.04 including Python 3.6 environment
- NVIDIA CUDA 10.2.89 including cuBLAS 10.2.2.89
- NVIDIA cuDNN 7.6.5
- NVIDIA NCCL 2.6.3 (optimized for NVLink™)
- APEX
- MLNX_OFED
- OpenMPI 3.1.4
- TensorBoard 2.1.0
- Nsight Compute 2019.5.0
- Nsight Systems 2020.1.1
- TensorRT 7.0.0
- DALI 0.19.0
- MAGMA 2.5.2
- Tensor Core optimized examples:
  - ResNeXt101-32x4d
  - SE-ResNext
  - TransformerXL
  - Jasper
  - BERT
  - Mask R-CNN
  - Tacotron 2 and WaveGlow v1.1

- SSD300 v1.1
- Neural Collaborative Filtering (NCF)
- ResNet50 v1.5
- GNMT v2

- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 1.0.4
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 20.03 is based on NVIDIA CUDA 10.2.89, which requires NVIDIA Driver release 440.33.01. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 396, 384.111+, 410, 418.xx or 440.30. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 20.03 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 20.03 is based on 1.5.0a0+8f84ded
- Latest version of DALI 0.19.0
- Performance improvements for elementwise operations
- Performance improvements for per-channel quantization
- Relaxation of cudnn batchnorm input shape requirements
- Ubuntu 18.04 with February 2020 updates

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.

- Transformer has been removed.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.03	18.04 16.04	NVIDIA CUDA 10.2.89	1.5.0a0+8f84ded	TensorRT 7.0.0
20.02			1.5.0a0+3bbb36e	
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

NVIDIA's Automatic Mixed Precision (AMP) for PyTorch is available in this container through a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Comprehensive guidance and examples demonstrating AMP for PyTorch can be found in the documentation.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub
- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- There is up to 5% performance drop on Transformer-XL mixed precision training in the 20.01 container compared to 19.11. Disabling the profiling executor at the beginning of your script might reduce this effect via:

```
torch._C._jit_set_profiling_executor(False)
torch._C._jit_set_profiling_mode(False)
```

- A workaround for the WaveGlow training regression from our past containers is to use a fake batch dimension when calculating the log determinant via `torch.logdet(W.unsqueeze(0).float()).squeeze()` as is done in this release.

Chapter 82. PyTorch Release 20.02

The NVIDIA container image for PyTorch, release 20.02, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 18.04 including Python 3.6 environment
- NVIDIA CUDA 10.2.89 including cuBLAS 10.2.2.89
- NVIDIA cuDNN 7.6.5
- NVIDIA NCCL 2.5.6 (optimized for NVLink™)
- APEX
- MLNX_OFED
- OpenMPI 3.1.4
- TensorBoard 2.1.0
- Nsight Compute 2019.5.0
- Nsight Systems 2020.1.1
- TensorRT 7.0.0
- DALI 0.18.0 Beta
- MAGMA 2.5.2
- Tensor Core optimized examples:
  - ResNeXt101-32x4d
  - SE-ResNext
  - TransformerXL
  - Jasper
  - BERT
  - Mask R-CNN
  - Tacotron 2 and WaveGlow v1.1

- SSD300 v1.1
- Neural Collaborative Filtering (NCF)
- ResNet50 v1.5
- GNMT v2

- Jupyter and JupyterLab:
  - Jupyter Client 5.3.4
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.3
  - JupyterLab 1.0.4
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 20.02 is based on NVIDIA CUDA 10.2.89, which requires NVIDIA Driver release 440.33.01. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 396, 384.111+, 410, 418.xx or 440.30. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 20.02 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 20.02 is based on PyTorch 1.4.0a0+a5b4d78 with a fix for wrong results in LU factorization using MAGMA<=2.5.1.
- Latest version of DALI 0.18.0 Beta
- Latest version of Nsight Systems 2020.1.1
- Latest version of Jupyter Notebook 6.0.3
- Ubuntu 18.04 with January 2020 updates
- Initial support for channel-last layout for convolutions
- Support for loop unrolling and vectorized loads and stores in TensorIterator
- Support for input activations with more than 2^{31} values

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.
- Transformer has been removed.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.02	18.04 16.04	NVIDIA CUDA 10.2.89	1.5.0a0+3bbb36e	TensorRT 7.0.0
20.01			1.4.0a0+a5b4d78	
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

NVIDIA's Automatic Mixed Precision (AMP) for PyTorch is available in this container through a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Comprehensive guidance and examples demonstrating AMP for PyTorch can be found in the documentation.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub
- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor

Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- There is up to 5% performance drop on Transformer-XL mixed precision training in the 20.01 container compared to 19.11. Disabling the profiling executor at the beginning of your script might reduce this effect via:

```
torch._C._jit_set_profiling_executor(False)
```

```
torch._C._jit_set_profiling_mode(False)
```

- A workaround for the WaveGlow training regression from our past containers is to use a fake batch dimension when calculating the log determinant via `torch.logdet(W.unsqueeze(0).float()).squeeze()` as is done in this release.

Chapter 83. PyTorch Release 20.01

The NVIDIA container image for PyTorch, release 20.01, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 18.04 including Python 3.6 environment
- NVIDIA CUDA 10.2.89 including cuBLAS 10.2.2.89
- NVIDIA cuDNN 7.6.5
- NVIDIA NCCL 2.5.6 (optimized for NVLink™)
- APEX
- MLNX_OFED
- OpenMPI 3.1.4
- TensorBoard 2.1.0
- Nsight Compute 2019.5.0
- Nsight Systems 2019.6.1
- TensorRT 7.0.0
- DALI 0.17.0 Beta
- MAGMA 2.5.2
- Tensor Core optimized examples:
  - ResNeXt101-32x4d
  - SE-ResNext
  - TransformerXL
  - Jasper
  - BERT
  - Mask R-CNN
  - Tacotron 2 and WaveGlow v1.1

- SSD300 v1.1
- Neural Collaborative Filtering (NCF)
- Transformer
- ResNet50 v1.5
- GNMT v2

- Jupyter and JupyterLab:
  - Jupyter Client 5.3.4
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.2
  - JupyterLab 1.0.4
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 20.01 is based on NVIDIA CUDA 10.2.89, which requires NVIDIA Driver release 440.33.01. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 396, 384.111+, 410, 418.xx or 440.30. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 20.01 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 20.01 is based on PyTorch 1.4.0a0+a5b4d78 with a fix for wrong results in LU factorization using MAGMA<=2.5.1.
- Latest version of TensorRT 7.0.0
- Latest version of DALI 0.17.0 Beta
- Latest version of MAGMA 2.5.2
- Ubuntu 18.04 with December 2019 updates

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.

NVIDIA PyTorch Container Versions

The following table shows what versions of Ubuntu, CUDA, PyTorch, and TensorRT are supported in each of the NVIDIA containers for PyTorch. For older container versions, refer to the Frameworks Support Matrix.

Container Version	Ubuntu	CUDA Toolkit	PyTorch	TensorRT
20.01	18.04 16.04	NVIDIA CUDA 10.2.89	1.4.0a0+a5b4d78	TensorRT 7.0.0
19.12 19.11			1.4.0a0+174e1ba	TensorRT 6.0.1
19.10		NVIDIA CUDA 10.1.243	1.3.0a0+24ae9b5	
19.09			1.2.0	
19.08			1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked	TensorRT 5.1.5

Automatic Mixed Precision (AMP)

NVIDIA's Automatic Mixed Precision (AMP) for PyTorch is available in this container through a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Comprehensive guidance and examples demonstrating AMP for PyTorch can be found in the documentation.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub
- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Transformer model. The Transformer model is based on the optimized implementation in Facebook's Fairseq NLP Toolkit and is built on top of PyTorch. The original version in the Fairseq project was developed using Tensor Cores, which provides significant training speedup. Our implementation improves the performance and is tested on a DGX-1V 16GB. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- There is up to 5% performance drop on Transformer-XL mixed precision training in the 20.01 container compared to 19.11. Disabling the profiling executor at the beginning of your script might reduce this effect via:

```
torch._C._jit_set_profiling_executor(False)
torch._C._jit_set_profiling_mode(False)
```

- A workaround for the WaveGlow training regression from our past containers is to use a fake batch dimension when calculating the log determinant via `torch.logdet(W.unsqueeze(0).float()).squeeze()` as is done in this release.
- The mixed-precision recipe for Transformer training might create unexpectedly NaN outputs. We recommend using FP32 or AMP with `opt_level='O0'` with the 20.01 container.

Chapter 84. PyTorch Release 19.12

The NVIDIA container image for PyTorch, release 19.12, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 18.04 including Python 3.6 environment
- NVIDIA CUDA 10.2.89 including cuBLAS 10.2.2.89
- NVIDIA cuDNN 7.6.5
- NVIDIA NCCL 2.5.6 (optimized for NVLink™)
- APEX
- MLNX_OFED
- OpenMPI 3.1.4
- TensorBoard 2.1.0
- Nsight Compute 2019.5.0
- Nsight Systems 2019.6.1
- TensorRT 6.0.1
- DALI 0.16.0 Beta
- Tensor Core optimized examples:
  - ResNeXt101-32x4d
  - SE-ResNext
  - TransformerXL
  - Jasper
  - BERT
  - Mask R-CNN
  - Tacotron 2 and WaveGlow v1.1
  - SSD300 v1.1

- Neural Collaborative Filtering (NCF)
- Transformer
- ResNet50 v1.5
- GNMT v2

- Jupyter and JupyterLab:
  - Jupyter Client 5.3.4
  - Jupyter Core 4.6.1
  - Jupyter Notebook 6.0.2
  - JupyterLab 1.0.4
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 19.12 is based on NVIDIA CUDA 10.2.89, which requires NVIDIA Driver release 440.30. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 396, 384.111+, 410, 418.xx or 440.30. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.12 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 19.12 is based on PyTorch 1.4.0a0+a5b4d78.
- Latest version of TensorBoard 2.1.0
- Latest version of DALI 0.16.0 Beta
- Latest version of Nsight Systems 2019.6.1
- Latest version of Jupyter Notebook 6.0.2
- Ubuntu 18.04 with November 2019 updates

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.

Automatic Mixed Precision (AMP)

NVIDIA's Automatic Mixed Precision (AMP) for PyTorch is available in this container through a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Comprehensive guidance and examples demonstrating AMP for PyTorch can be found in the documentation.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub

- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Transformer model. The Transformer model is based on the optimized implementation in Facebook's Fairseq NLP Toolkit and is built on top of PyTorch. The original version

in the Fairseq project was developed using Tensor Cores, which provides significant training speedup. Our implementation improves the performance and is tested on a DGX-1V 16GB. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- There is up to 70% performance drop on Waveglow training in the 19.12 container compared to 19.10; due to a magma-CUDA bug recreating cuBLAS handles. You can minimize this regression by setting the `CUBLAS_WORKSPACE_CONFIG=:16:8` environment variable before running the Waveglow code.
- There is up to 20% performance drop for SSD training on Pascal GPUs in the 19.12 container compared to 19.11.
- There is up to 8% performance drop on BERT Large mixed-precision training in the 19.12 container compared to 19.10.
- The mixed-precision recipe for Transformer training might create unexpectedly NaN outputs. We recommend to use FP32 or AMP with `opt_level='O0'` with the 19.12 container.

Chapter 85. PyTorch Release 19.11

The NVIDIA container image for PyTorch, release 19.11, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 18.04 including Python 3.6 environment
- NVIDIA CUDA 10.2.89 including cuBLAS 10.2.2.89
- NVIDIA cuDNN 7.6.5
- NVIDIA NCCL 2.5.6 (optimized for NVLink™)
- APEX
- MLNX_OFED
- OpenMPI 3.1.4
- TensorBoard 2.0.1
- Nsight Compute 2019.5.0
- Nsight Systems 2019.5.2
- TensorRT 6.0.1
- DALI 0.15.0 Beta
- Tensor Core optimized examples:
  - ResNeXt101-32x4d
  - SE-ResNext
  - TransformerXL
  - Jasper
  - BERT
  - Mask R-CNN
  - Tacotron 2 and WaveGlow v1.1
  - SSD300 v1.1

  - Neural Collaborative Filtering (NCF)
  - Transformer
  - ResNet50 v1.5
  - GNMT v2
- Jupyter and JupyterLab:
  - Jupyter Client 5.3.4
  - Jupyter Core 4.6.1
  - Jupyter Notebook 5.7.8
  - JupyterLab 1.0.4
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 19.11 is based on NVIDIA CUDA 10.2.89, which requires NVIDIA Driver release 440.30. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 396, 384.111+, 410 or 418.xx. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.11 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 19.11 is based on PyTorch 1.4.0a0+174e1ba with cherry-picked fixes for TensorIterator, LayNerNorm as well as NCCL 2.5.
- Latest version of NVIDIA CUDA 10.2.89 including cuBLAS 10.2.2.89
- Latest version of TensorBoard 2.0.1
- Latest version of NVIDIA cuDNN 7.6.5
- Latest version of NVIDIA NCCL 2.5.6
- Latest version of Nsight Compute 2019.5.0
- Latest version of Nsight Systems 2019.5.2
- Latest version of DALI 0.15.0 Beta
- Latest versions of Jupyter Client 5.3.4 and Jupyter Core 4.6.1

- Added a TransformerXL Tensor Core optimized example
- Ubuntu 18.04 with October 2019 updates

Announcements

- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.

Automatic Mixed Precision (AMP)

NVIDIA's Automatic Mixed Precision (AMP) for PyTorch is available in this container through a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Comprehensive guidance and examples demonstrating AMP for PyTorch can be found in the documentation.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub and NVIDIA GPU Cloud (NGC) focus on achieving the best performance and convergence from NVIDIA Volta tensor cores by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta and Turing, therefore you can get results much faster than training without Tensor Cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- TransformerXL model. Transformer-XL is a transformer-based language model with a segment-level recurrence and a novel relative positional encoding. Enhancements introduced in Transformer-XL help capture better long-term dependencies by

attending to tokens from multiple previous segments. Our implementation is based on the codebase published by the authors of the Transformer-XL paper. Our implementation uses modified model architecture hyperparameters. Our modifications were made to achieve better hardware utilization and to take advantage of Tensor Cores. his model script is available on GitHub

- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao,

Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- Transformer model. The Transformer model is based on the optimized implementation in Facebook's Fairseq NLP Toolkit and is built on top of PyTorch. The original version in the Fairseq project was developed using Tensor Cores, which provides significant training speedup. Our implementation improves the performance and is tested on a DGX-1V 16GB. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- The performance of Mask R-CNN in FP32 precision is up to 14% slower in the 19.11 container compared to the 19.06 release. For best performance on Mask R-CNN, it is recommended to use automatic mixed precision training. This can easily be done using the float16 option with the MaskRCNN example included in this container.
- There is up to 66% performance drop on WaveGlow training in the 19.11 container compared to 19.10.
- The mixed-precision recipe for Transformer training might create unexpectedly NaN outputs. We recommend to use FP32 or AMP with `opt_level='O0'` with the 19.11 container.
- There is a 3-15% performance drop on Tacotron2 inference in the 19.11 container compared to 19.09.

Chapter 86. PyTorch Release 19.10

The NVIDIA container image for PyTorch, release 19.10, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 18.04 including Python 3.6 environment
- NVIDIA CUDA 10.1.243 including cuBLAS 10.2.1.243
- NVIDIA cuDNN 7.6.4
- NVIDIA NCCL 2.4.8 (optimized for NVLink™)
- APEX
- MLNX_OFED
- OpenMPI 3.1.4
- TensorBoard 2.0.0
- Nsight Compute 2019.4.0
- Nsight Systems 2019.5.1
- TensorRT 6.0.1
- DALI 0.14.0 Beta
- Tensor Core optimized examples:
  - ResNeXt101-32x4d
  - SE-ResNext
  - Jasper
  - BERT
  - Mask R-CNN
  - Tacotron 2 and WaveGlow v1.1
  - SSD300 v1.1
  - Neural Collaborative Filtering (NCF)

  - Transformer
  - ResNet50 v1.5
  - GNMT v2
- Jupyter and JupyterLab:
  - Jupyter Client 5.3.3
  - Jupyter Core 4.5.0
  - Jupyter Notebook 5.7.8
  - JupyterLab 1.0.4
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 19.10 is based on NVIDIA CUDA 10.1.243, which requires NVIDIA Driver release 418.xx. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 396, 384.111+ or 410. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.10 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 19.10 is based on PyTorch 1.3.0a0+24ae9b5.
- Latest version of NVIDIA cuDNN 7.6.4
- Latest version of TensorBoard 2.0.0
- Latest version of DALI 0.14.0 Beta
- Latest version of Nsight Systems 2019.5.1
- Latest version of Jupyter Client 5.3.3
- Added the ResNeXt101-32x4d and SE-ResNext Tensor Core optimized model examples.
- Ubuntu 18.04 with September 2019 updates

Automatic Mixed Precision (AMP)

NVIDIA's Automatic Mixed Precision (AMP) for PyTorch is available in this container through a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Comprehensive guidance and examples demonstrating AMP for PyTorch can be found in the documentation.

For more information about AMP, see the Training With Mixed Precision Guide.

Tensor Core Examples

The tensor core examples provided in GitHub focus on achieving the best performance and convergence by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- ResNeXt101-32x4d model. The ResNeXt101-32x4d is a model introduced in the Aggregated Residual Transformations for Deep Neural Networks paper. It is based on regular ResNet model, substituting 3x3 convolutions inside the bottleneck block for 3x3 grouped convolutions. This model script is available on GitHub.
- SE-ResNext model. The SE-ResNeXt101-32x4d is a ResNeXt101-32x4d model with added Squeeze-and-Excitation (SE) module introduced in the Squeeze-and-Excitation Networks paper. This model script is available on GitHub.
- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model

is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- Transformer model. The Transformer model is based on the optimized implementation in Facebook's Fairseq NLP Toolkit and is built on top of PyTorch. The original version in the Fairseq project was developed using Tensor Cores, which provides significant training speedup. Our implementation improves the performance and is tested on a DGX-1V 16GB. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- Performance of Mask R-CNN in FP32 precision is up to 20% slower in the 19.10 container compared to the 19.06 release. For best performance on Mask R-CNN, it is recommended to use automatic mixed precision training. This can easily be done using the float16 option with the MaskRCNN example included in this container.
- There is a 15-20% performance drop on WaveGlow inference in the 19.10 container compared to 19.08 using automatic mixed precision (AMP) with V100 compared to previous releases. To workaround this issue, install cuDNN 7.6.2 or use the 19.08 container.
- The mixed-precision recipe for Transformer training might create unexpectedly NaN outputs. We recommend to use FP32 or AMP with `opt_level='O0'` with the 19.10 container.
- There is a 3-15% performance drop on Tacotron2 inference in the 19.10 container compared to the previous release.

Chapter 87. PyTorch Release 19.09

The NVIDIA container image for PyTorch, release 19.09, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 18.04 including Python 3.6 environment
- NVIDIA CUDA 10.1.243 including cuBLAS 10.2.1.243
- NVIDIA cuDNN 7.6.3
- NVIDIA NCCL 2.4.8 (optimized for NVLink™)
- APEX
- MLNX_OFED
- OpenMPI 3.1.4
- TensorBoard 1.14.0+nv
- Nsight Compute 2019.4.0
- Nsight Systems 2019.4.2
- TensorRT 6.0.1
- DALI 0.12.0 Beta
- Tensor Core optimized examples:
  - Jasper
  - BERT
  - Mask R-CNN
  - Tacotron 2 and WaveGlow v1.1
  - SSD300 v1.1
  - Neural Collaborative Filtering (NCF)
  - Transformer
  - ResNet50 v1.5

- GNMT v2
- Jupyter and JupyterLab:
  - Jupyter Client 5.3.1
  - Jupyter Core 4.5.0
  - Jupyter Notebook 5.7.8
  - JupyterLab 1.0.4
  - JupyterLab Server 1.0.6
  - Jupyter-TensorBoard

Driver Requirements

Release 19.09 is based on NVIDIA CUDA 10.1.243, which requires NVIDIA Driver release 418.xx. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 396, 384.111+ or 410. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.09 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 19.09 is based on PyTorch 1.2.0.
- Latest version of NVIDIA cuDNN 7.6.3
- Latest versions of Nsight Compute 2019.4.0 and Nsight Systems 2019.4.2
- Latest version of TensorRT 6.0.1
- Latest version of JupyterLab Server 1.0.6
- Ubuntu 18.04 with August 2019 updates

Tensor Core Examples

These examples focus on achieving the best performance and convergence from NVIDIA Volta Tensor Cores by using the latest deep learning example networks for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against

each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- Transformer model. The Transformer model is based on the optimized implementation in Facebook's Fairseq NLP Toolkit and is built on top of PyTorch. The original version in the Fairseq project was developed using Tensor Cores, which provides significant training speedup. Our implementation improves the performance and is tested on a DGX-1V 16GB. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Automatic Mixed Precision (AMP)

NVIDIA's Automatic Mixed Precision (AMP) for PyTorch is available in this container through a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Comprehensive guidance and examples demonstrating AMP for PyTorch can be found in the documentation.

For more information about AMP, see the Training With Mixed Precision Guide.

Known Issues

- Performance of Mask R-CNN in FP32 precision is up to 20% slower in the 19.07 container compared to the previous release. For best performance on Mask R-CNN, it is recommended to use automatic mixed precision training. This can easily be done using the float16 option with the MaskRCNN example included in this container.
- Due to recent changes on batch norm multiplier initialization (PyTorch commit: c60465873c5cf8f1a36da39f7875224d4c48d7ca), all batch norm multiplier is initialized as constant 1, instead of uniformly distributed between 0 and 1, as it was previously. This has caused accuracy issue for our TACOTRON2 model. If similar accuracy regression is observed during an update from 19.06 to 19.08, we recommend to re-initialize the batch norm multiplier using uniformed distribution. This could be done by passing your model to the following function:

```
def init_bn(module):
    if isinstance(module, torch.nn.modules.batchnorm._BatchNorm):
     if module.affine:
         module.weight.data.uniform_()
    for child in module.children():
     init_bn(child)
```

- There is a 34-60% performance drop on WaveGlow training in the 19.09 container on 16 GPU systems using mixed precision training compared to previous releases.
- There is a 15-20% performance drop on WaveGlow inference in the 19.09 container using automatic mixed precision (AMP) with V100 compared to previous releases. To workaround this issue, install cuDNN 7.6.2 or use the 19.08 container.
- Nsight Compute is currently located in `/opt/nvidia/nsight-compute/2019.4.0`, while Nsight Systems can be found in `/usr/local/cuda-10.1/NsightSystems-cli-2019.4.2/bin/nsys`.

Chapter 88. PyTorch Release 19.08

The NVIDIA container image for PyTorch, release 19.08, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 18.04 including Python 3.6 environment
- NVIDIA CUDA 10.1.243 including cuBLAS 10.2.1.243
- NVIDIA cuDNN 7.6.2
- NVIDIA NCCL 2.4.8 (optimized for NVLink™)
- APEX
- MLNX_OFED +4.0
- OpenMPI 3.1.4
- TensorBoard 1.14.0+nv
- Nsight Compute 10.1.168
- Nsight Systems 2019.3.7.9
- TensorRT 5.1.5
- DALI 0.12.0 Beta
- Tensor Core optimized examples:
  - Jasper
  - BERT
  - Mask R-CNN
  - Tacotron 2 and WaveGlow v1.1
  - SSD300 v1.1
  - Neural Collaborative Filtering (NCF)
  - Transformer
  - ResNet50 v1.5

  - GNMT v2
- Jupyter and JupyterLab:
  - Jupyter Client 5.3.1
  - Jupyter Core 4.5.0
  - Jupyter Notebook 5.7.8
  - JupyterLab 1.0.4
  - JupyterLab Server 1.0.0
  - Jupyter-TensorBoard

Driver Requirements

Release 19.08 is based on NVIDIA CUDA 10.1.243, which requires NVIDIA Driver release 418.87. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384.111+ or 410. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.08 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 19.08 is based on PyTorch 1.2.0a0 including upstream commits up through commit 9130ab38 from July 31, 2019 as well as a cherry-picked performance fix 9462ca29.
- Latest version of NVIDIA CUDA 10.1.243 including cuBLAS 10.2.1.243
- Latest version of NVIDIA cuDNN 7.6.2
- Latest version of NVIDIA NCCL 2.4.8
- Latest version of DALI 0.12.0 Beta
- Latest version of OpenMPI 3.1.4
- Latest version of Nsight Systems 2019.3.7.9
- Latest version of MLNX_OFED +4.0
- Added a Jasper Tensor Core model script example
- Ubuntu 18.04 with July 2019 updates

Tensor Core Examples

These examples focus on achieving the best performance and convergence from NVIDIA Volta Tensor Cores by using the latest deep learning example networks for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- Jasper model. This repository provides an implementation of the Jasper model in PyTorch from the paper Jasper: An End-to-End Convolutional Neural Acoustic Model. The Jasper model is an end-to-end neural acoustic model for automatic speech recognition (ASR) that provides near state-of-the-art results on LibriSpeech among end-to-end ASR models without any external data. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- NCF model. The Neural Collaborative Filtering (NCF) model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- Transformer model. The Transformer model is based on the optimized implementation in Facebook's Fairseq NLP Toolkit and is built on top of PyTorch. The original version in the Fairseq project was developed using Tensor Cores, which provides significant training speedup. Our implementation improves the performance and is tested on a DGX-1V 16GB. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Automatic Mixed Precision (AMP)

NVIDIA's Automatic Mixed Precision (AMP) for PyTorch is available in this container through a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Comprehensive guidance and examples demonstrating AMP for PyTorch can be found in the documentation.

For more information about AMP, see the Training With Mixed Precision Guide.

Known Issues

- Performance of Mask R-CNN in FP32 precision is up to 20% slower in the 19.07 container compared to the previous release. For best performance on Mask R-CNN,

it is recommended to use automatic mixed precision training. This can easily be done using the float16 option with the MaskRCNN example included in this container.

- Due to recent changes on batch norm multiplier initialization (PyTorch commit: c60465873c5cf8f1a36da39f7875224d4c48d7ca), all batch norm multiplier is initialized as constant 1, instead of uniformly distributed between 0 and 1, as it was previously. This has caused accuracy issue for our TACOTRON2 model. If similar accuracy regression is observed during an update from 19.06 to 19.08, we recommend to re-initialize the batch norm multiplier using uniformed distribution. This could be done by passing your model to the following function:

```
def init_bn(module):
    if isinstance(module, torch.nn.modules.batchnorm._BatchNorm):
     if module.affine:
         module.weight.data.uniform_()
    for child in module.children():
     init_bn(child)
```

Chapter 89. PyTorch Release 19.07

The NVIDIA container image for PyTorch, release 19.07, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 18.04 including Python 3.6 environment
- NVIDIA CUDA 10.1.168 including cuBLAS 10.2.0.168
- NVIDIA cuDNN 7.6.1
- NVIDIA NCCL 2.4.7 (optimized for NVLink™)
- APEX
- MLNX_OFED +3.4
- OpenMPI 3.1.3
- TensorBoard 1.14.0+nv
- TensorRT 5.1.5
- DALI 0.11.0 Beta
- Tensor Core optimized examples:
  - BERT
  - Mask R-CNN
  - Tacotron 2 and WaveGlow v1.1
  - SSD300 v1.1
  - Neural Collaborative Filtering (NCF)
  - Transformer
  - ResNet50 v1.5
  - GNMT v2
- Jupyter and JupyterLab:

- Jupyter Client 5.3.1
- Jupyter Core 4.5.0
- Jupyter Notebook 5.7.8
- JupyterLab 1.0.2
- JupyterLab Server 1.0.0
- Jupyter-TensorBoard

Driver Requirements

Release 19.07 is based on NVIDIA CUDA 10.1.168, which requires NVIDIA Driver release 418.67. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384.111+ or 410. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.07 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 19.07 is based on PyTorch 1.2.0a0 including upstream commits up through commit f6aac41 from June 19, 2019.
- Latest version of NVIDIA cuDNN 7.6.1
- Latest version of MLNX_OFED +3.4
- Added TensorBoard 1.14.0+nv to the container.
- Latest versions of Jupyter Client 5.3.1, Jupyter Core 4.5.0, JupyterLab 1.0.2 and JupyterLab Server 1.0.0, including Jupyter-TensorBoard integration.
- Latest version of DALI 0.11.0 Beta
- Latest version of Ubuntu 18.04

Tensor Core Examples

These examples focus on achieving the best performance and convergence from NVIDIA Volta Tensor Cores by using the latest deep learning example networks for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against

each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.
- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy.
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper.
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model.
- NCF model. The Neural Collaborative Filtering (NCF) focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper.
- Transformer model. The Transformer model is based on the optimized implementation in Facebook's Fairseq NLP Toolkit and is built on top of PyTorch. The original version in the Fairseq project was developed using Tensor Cores, which provides significant training speedup. Our implementation improves the performance and is tested on a DGX-1V 16GB.
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a slightly modified version of the original ResNet50 v1 model that trains to a greater accuracy.
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper.

Automatic Mixed Precision (AMP)

NVIDIA's Automatic Mixed Precision (AMP) for PyTorch is available in this container through a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Comprehensive guidance and examples demonstrating AMP for PyTorch can be found in the documentation.

For more information about AMP, see the Training With Mixed Precision Guide.

Known Issues

- Performance of Mask R-CNN in FP32 precision is up to 20% slower in the 19.07 container compared to the previous release. For best performance on Mask R-CNN, it is recommended to use automatic mixed precision training. This can easily be done using the float16 option with the MaskRCNN example included in this container.
- Due to recent changes on batch norm multiplier initialization (PyTorch commit: c60465873c5cf8f1a36da39f7875224d4c48d7ca), all batch norm multiplier is initialized as constant 1, instead of uniformly distributed between 0 and 1, as it was previously. This has caused accuracy issue for our TACOTRON2 model. If similar accuracy regression is observed during an update from 19.06 to 19.07, we recommend to re-initialize the batch norm multiplier using uniformed distribution. This could be done by passing your model to the following function:

```
def init_bn(module):
    if isinstance(module, torch.nn.modules.batchnorm._BatchNorm):
     if module.affine:
         module.weight.data.uniform_()
    for child in module.children():
     init_bn(child)
```

Chapter 90. PyTorch Release 19.06

The NVIDIA container image for PyTorch, release 19.06, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 10.1.168 including cuBLAS 10.2.0.168
- NVIDIA cuDNN 7.6.0
- NVIDIA NCCL 2.4.7 (optimized for NVLink™)
- APEX
- OpenMPI 3.1.3
- TensorRT 5.1.5
- DALI 0.10.0 Beta
- Tensor Core optimized examples:
  - BERT
  - Mask R-CNN
  - Tacotron 2 and WaveGlow v1.1
  - SSD300 v1.1
  - Neural Collaborative Filtering (NCF)
  - Transformer
  - ResNet50 v1.5
  - GNMT v2
- Jupyter and JupyterLab:
  - Jupyter Client 5.2.4
  - Jupyter Core 4.4.0

- Jupyter Notebook 5.7.8
- JupyterLab 0.35.6
- JupyterLab Server 0.2.0

Driver Requirements

Release 19.06 is based on NVIDIA CUDA 10.1.168, which requires NVIDIA Driver release 418.xx. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384.111+ or 410. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.06 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 19.06 is based on PyTorch 1.1.0commit 0885dd28 from May 28, 2019
- Added the BERT Tensor Core example
- Latest version of NVIDIA CUDA 10.1.168 including cuBLAS 10.2.0.168
- Latest version of NVIDIA NCCL 2.4.7
- Latest version of DALI 0.10.0 Beta
- Latest version of JupyterLab 0.35.6
- Ubuntu 16.04 with May 2019 updates (see Announcements)

Tensor Core Examples

These examples focus on achieving the best performance and convergence from NVIDIA Volta Tensor Cores by using the latest deep learning example networks for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-

art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on theBERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA's implementation of BERT is an optimized version of the Hugging Face implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

- Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy.
- Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper.
- SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model.
- NCF model. The Neural Collaborative Filtering (NCF) focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper.
- Transformer model. The Transformer model is based on the optimized implementation in Facebook's Fairseq NLP Toolkit and is built on top of PyTorch. The original version in the Fairseq project was developed using Tensor Cores, which provides significant training speedup. Our implementation improves the performance and is tested on a DGX-1V 16GB.
- ResNet50 v1.5 model. The ResNet50 v1.5 model is a slightly modified version of the original ResNet50 v1 model that trains to a greater accuracy.
- GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper.

Automatic Mixed Precision (AMP)

NVIDIA's Automatic Mixed Precision (AMP) for PyTorch is available in this container through a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 3 lines of Python to an existing FP32 (default) script. Amp will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Comprehensive guidance and examples demonstrating AMP for PyTorch can be found in the documentation.

For more information about AMP, see the Training With Mixed Precision Guide.

Announcements

In the next release, we will no longer support Ubuntu 16.04. Release 19.07 will instead support Ubuntu 18.04.

Known Issues

- There is a known issue when running certain tests in PyTorch 19.06 on systems with Skylake CPUs, such as DGX-2, that is due to OpenBLAS version 0.3.6. If you are impacted, run:

```
/opt/conda/bin/conda install openblas!=0.3.6
```

Chapter 91. PyTorch Release 19.05

The NVIDIA container image for PyTorch, release 19.05, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 10.1 Update 1 including cuBLAS 10.1 Update 1
- NVIDIA cuDNN 7.6.0
- NVIDIA NCCL 2.4.6 (optimized for NVLink™)
- APEX
- OpenMPI 3.1.3
- TensorRT 5.1.5
- DALI 0.9.1 Beta
- Tensor Core optimized examples:
  - Mask R-CNN
  - Tacotron 2 and WaveGlow v1.1
  - SSD300 v1.1
  - Neural Collaborative Filtering (NCF)
  - Transformer
  - ResNet50 v1.5
  - GNMT v2
- Jupyter and JupyterLab:
  - Jupyter Client 5.2.4
  - Jupyter Core 4.4.0
  - JupyterLab 0.35.4

- JupyterLab Server 0.2.0

Driver Requirements

Release 19.05 is based on CUDA 10.1 Update 1, which requires NVIDIA Driver release 418.xx. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384.111+ or 410. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.05 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 19.05 is based on PyTorch 1.0.1 commit 828a6a3b from March 31, 2019
- Latest version of NVIDIA CUDA 10.1 Update 1 including cuBLAS 10.1 Update 1
- Latest version of NVIDIA cuDNN 7.6.0
- Latest version of TensorRT 5.1.5
- Latest version of DALI 0.9.1 Beta
- Ubuntu 16.04 with April 2019 updates

Tensor Core Examples

These examples focus on achieving the best performance and convergence from NVIDIA Volta Tensor Cores by using the latest deep learning example networks for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- An implementation of the Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy.

- An implementation of the Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper.
- An implementation of the SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model.
- An implementation of the Neural Collaborative Filtering (NCF) model. The NCF model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper.
- An implementation of the Transformer model architecture. The Transformer model is based on the optimized implementation in Facebook's Fairseq NLP Toolkit and is built on top of PyTorch. The original version in the Fairseq project was developed using Tensor Cores, which provides significant training speedup. Our implementation improves the performance and is tested on a DGX-1V 16GB.
- An implementation of the ResNet50 model. The ResNet50 v1.5 model is a slightly modified version of the original ResNet50 v1 model that trains to a greater accuracy.
- An implementation of the GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper.

Automatic Mixed Precision (AMP)

NVIDIA's Automatic Mixed Precision (AMP) for PyTorch is available in this container through a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 2 lines of Python to an existing FP32 (default) script. AMP will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Comprehensive guidance and examples demonstrating AMP for PyTorch can be found in the documentation.

For more information about AMP, see the Training With Mixed Precision Guide.

Known Issues

- Persistent batch normalization kernels are enabled by default in this build. This will provide a performance boost to many networks, but in rare cases may cause a network to fail to train properly. We expect to address this in the 19.06 container.

Chapter 92. PyTorch Release 19.04

The NVIDIA container image for PyTorch, release 19.04, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 10.1.105 including cuBLAS 10.1.0.105
- NVIDIA cuDNN 7.5.0
- NVIDIA NCCL 2.4.6 (optimized for NVLink™)
- APEX
- OpenMPI 3.1.3
- TensorRT 5.1.2
- DALI 0.8.1 Beta
- Tensor Core optimized examples:
  - Mask R-CNN
  - Tacotron 2 and WaveGlow v1.1
  - SSD300 v1.1
  - Neural Collaborative Filtering (NCF)
  - Transformer
  - ResNet50 v1.5
  - GNMT v2
- Jupyter and JupyterLab:
  - Jupyter Client 5.2.4
  - Jupyter Core 4.4.0
  - JupyterLab 0.35.4

- JupyterLab Server 0.2.0

Driver Requirements

Release 19.04 is based on CUDA 10.1, which requires NVIDIA Driver release 418.xx.x +. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384.111+ or 410. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.04 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 19.04 is based on PyTorch 1.0.1 commit 9eb0f43 from March 28, 2019
- Latest version of NVIDIA NCCL 2.4.6
- Latest version of DALI 0.8.1 Beta
- Latest version of cuBLAS 10.1.0.105
- Added the Mask R-CNN Tensor Core example
- Ubuntu 16.04 with March 2019 updates

Tensor Core Examples

These examples focus on achieving the best performance and convergence from NVIDIA Volta Tensor Cores by using the latest deep learning example networks for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- An implementation of the Mask R-CNN model. Mask R-CNN is a convolution based neural network for the task of object instance segmentation. The paper describing the model can be found here. NVIDIA's Mask R-CNN model is an optimized version of Facebook's implementation, leveraging mixed precision arithmetic using Tensor Cores on NVIDIA Tesla V100 GPUs for 1.3x faster training time while maintaining target accuracy.

- An implementation of the Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper.
- An implementation of the SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model.
- An implementation of the Neural Collaborative Filtering (NCF) model. The NCF model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper.
- An implementation of the Transformer model architecture. The Transformer model is based on the optimized implementation in Facebook's Fairseq NLP Toolkit and is built on top of PyTorch. The original version in the Fairseq project was developed using Tensor Cores, which provides significant training speedup. Our implementation improves the performance and is tested on a DGX-1V 16GB.
- An implementation of the ResNet50 model. The ResNet50 v1.5 model is a slightly modified version of the original ResNet50 v1 model that trains to a greater accuracy.
- An implementation of the GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper.

Automatic Mixed Precision (AMP)

NVIDIA's Automatic Mixed Precision (AMP) for PyTorch is available in this container through a preinstalled release of Apex. AMP enables users to try mixed precision training by adding only 2 lines of Python to an existing FP32 (default) script. AMP will choose an optimal set of operations to cast to FP16. FP16 operations require 2X reduced memory bandwidth (resulting in a 2X speedup for bandwidth-bound operations like most pointwise ops) and 2X reduced memory storage for intermediates (reducing the overall memory consumption of your model). Additionally, GEMMs and convolutions with FP16 inputs can run on Tensor Cores, which provide an 8X increase in computational throughput over FP32 arithmetic.

Comprehensive guidance and examples demonstrating AMP for PyTorch can be found in the documentation.

For more information about AMP, see the Training With Mixed Precision Guide.

Known Issues

- Persistent batch normalization kernels are enabled by default in this build. This will provide a performance boost to many networks, but in rare cases may cause a network to fail to train properly. We expect to address this in the 19.05 container.

Chapter 93. PyTorch Release 19.03

The NVIDIA container image for PyTorch, release 19.03, is available on NGC.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 10.1.105 including cuBLAS 10.1.105
- NVIDIA cuDNN 7.5.0
- NVIDIA NCCL 2.4.3 (optimized for NVLink™)
- APEX
- OpenMPI 3.1.3
- TensorRT 5.1.2
- DALI 0.7 Beta
- Tensor Core optimized examples:
  - Tacotron 2 and WaveGlow v1.1
  - SSD300 v1.1
  - Neural Collaborative Filtering (NCF)
  - Transformer
  - ResNet50 v1.5
  - GNMT v2
- Jupyter and JupyterLab:
  - Jupyter Client 5.2.4
  - Jupyter Core 4.4.0
  - JupyterLab 0.35.4
  - JupyterLab Server 0.2.0

Driver Requirements

Release 19.03 is based on CUDA 10.1, which requires NVIDIA Driver release 418.xx+. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384.111+ or 410. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.03 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 19.03 is based on PyTorch commit 81e025d from March 9th, 2019
- Latest version of NVIDIA CUDA 10.1.105 including cuBLAS 10.1.105
- Latest version of NVIDIA cuDNN 7.5.0
- Latest version of NVIDIA NCCL 2.4.3
- Latest version of DALI 0.7 Beta
- Latest version of TensorRT 5.1.2
- Added the Tacotron 2 and WaveGlow v1.1 and SSD300 v1.1 Tensor Core examples
- Ubuntu 16.04 with February 2019 updates

Tensor Core Examples

These examples focus on achieving the best performance and convergence from NVIDIA Volta Tensor Cores by using the latest deep learning example networks for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- An implementation of the Tacotron 2 and WaveGlow v1.1 model. This text-to-speech (TTS) system is a combination of two neural network models: a modified Tacotron 2 model from the Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions paper and a flow-based neural network model from the WaveGlow: A Flow-based Generative Network for Speech Synthesis paper.

- An implementation of the SSD300 v1.1 model. The SSD300 v1.1 model is based on the SSD: Single Shot MultiBox Detector paper. The main difference between this model and the one described in the paper is in the backbone. Specifically, the VGG model is obsolete and is replaced by the ResNet50 model.
- An implementation of the Neural Collaborative Filtering (NCF) model. The NCF model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper.
- An implementation of the Transformer model architecture. The Transformer model is based on the optimized implementation in Facebook's Fairseq NLP Toolkit and is built on top of PyTorch. The original version in the Fairseq project was developed using Tensor Cores, which provides significant training speedup. Our implementation improves the performance and is tested on a DGX-1V 16GB.
- An implementation of the ResNet50 model. The ResNet50 v1.5 model is a slightly modified version of the original ResNet50 v1 model that trains to a greater accuracy.
- An implementation of the GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper.

Known Issues

- Persistent batch normalization kernels have been disabled due to a known bug during validation. Batch normalization provides correct results and work as expected from users, however, this may cause up to 10% regression in time to solution performance on networks using batch normalization.
- If using or upgrading to a 3-part-version driver, for example, a driver that takes the format of `xxx.yy.zz`, you will receive a `Failed to detect NVIDIA driver version.` message. This is due to a known bug in the entry point script's parsing of 3-part driver versions. This message is non-fatal and can be ignored. This will be fixed in the 19.04 release.

Chapter 94. PyTorch Release 19.02

The NVIDIA container image for PyTorch, release 19.02, is available.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in Conda™ default environment (`/opt/conda/lib/python3.6/site-packages/torch/`) in the container image.

The container also includes the following:

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 10.0.130 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 10.0.130
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.4.2
- NVIDIA Collective Communications Library (NCCL) 2.3.7 (optimized for NVLink™)
- APEX
- OpenMPI 3.1.3
- TensorRT 5.0.2
- DALI 0.6.1 Beta
- Tensor Core optimized examples:
  - Neural Collaborative Filtering (NCF)
  - Transformer
  - ResNet50 v1.5
  - GNMT v2
- Jupyter and JupyterLab:
  - Jupyter Client 5.2.4
  - Jupyter Core 4.4.0
  - JupyterLab 0.35.4
  - JupyterLab Server 0.2.0

Driver Requirements

Release 19.02 is based on CUDA 10, which requires NVIDIA Driver release 410.xx. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.02 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 19.02 is based on PyTorch Version 1.1.0a0+c42431b.
- Latest version of DALI 0.6.1 Beta
- Added Jupyter and JupyterLab software in our packaged container.
- Latest version of jupyter_client 5.2.4
- Latest version of jupyter_core 4.4.0
- Ubuntu 16.04 with January 2019 updates

Tensor Core Examples

These examples focus on achieving the best performance and convergence from NVIDIA Volta Tensor Coress by using the latest deep learning example networks for training. This container includes the following Tensor Core examples.

- An implementation of the Neural Collaborative Filtering (NCF) model. The NCF model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper.

- An implementation of the Transformer model architecture. The Transformer model is based on the optimized implementation in Facebook's Fairseq NLP Toolkit and is built on top of PyTorch. The original version in the Fairseq project was developed using Tensor Cores, which provides significant training speedup. Our implementation improves the performance and is tested on a DGX-1V 16GB.

- An implementation of the ResNet50 model. The ResNet50 v1.5 model is a slightly modified version of the original ResNet50 v1 model that trains to a greater accuracy.
- An implementation of the GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper.

Known Issues

- Persistent batch normalization kernels have been disabled due to a known bug during validation. Batch normalization provides correct results and work as expected from users, however, this may cause up to 10% regression in time to solution performance on networks using batch normalization.
- If using or upgrading to a 3-part-version driver, for example, a driver that takes the format of `xxx.yy.zz`, you will receive a `Failed to detect NVIDIA driver version.` message. This is due to a known bug in the entry point script's parsing of 3-part driver versions. This message is non-fatal and can be ignored. This will be fixed in the 19.04 release.

Chapter 95. PyTorch Release 19.01

The NVIDIA container image for PyTorch, release 19.01, is available.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in the `pytorch-py3.6` Conda™ environment in the container image.

The container also includes the following:

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 10.0.130 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 10.0.130
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.4.2
- NCCL 2.3.7 (optimized for NVLink™)
- OpenMPI 3.1.3
- Caffe2
- TensorRT 5.0.2
- DALI 0.6 Beta
- Tensor Core optimized examples:
  - Neural Collaborative Filtering (NCF)
  - Transformer
  - ResNet50 v1.5
  - GNMT v2

Driver Requirements

Release 19.01 is based on CUDA 10, which requires NVIDIA Driver release 410.xx. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.01 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 19.01 is based on PyTorch v1.0.0 with up-to-date features.
- Latest version of DALI 0.6 Beta
- Latest version of NVIDIA cuDNN 7.4.2
- Latest version of OpenMPI 3.1.3
- Added the Neural Collaborative Filtering (NCF) and Transformer Tensor Core examples.
- Ubuntu 16.04 with December 2018 updates

Tensor Core Examples

These examples focus on achieving the best performance and convergence from NVIDIA Volta Tensor Cores by using the latest deep learning example networks for training. This container includes the following Tensor Core examples.

- An implementation of the Neural Collaborative Filtering (NCF) model. The NCF model focuses on providing recommendations, also known as collaborative filtering; with implicit feedback. The training data for this model should contain binary information about whether a user interacted with a specific item. NCF was first described by Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua in the Neural Collaborative Filtering paper.
- An implementation of the Transformer model architecture. The Transformer model is based on the optimized implementation in Facebook's Fairseq NLP Toolkit and is built on top of PyTorch. The original version in the Fairseq project was developed using Tensor Cores, which provides significant training speedup. Our implementation improves the performance and is tested on a DGX-1V 16GB.
- An implementation of the ResNet50 model. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model.
- An implementation of the GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper.

Known Issues

- Persistent batch normalization kernels have been disabled due to a known bug during validation. Batch normalization provides correct results and work as expected from users, however, this may cause up to 10% regression in time to solution performance on networks using batch normalization.
- If using or upgrading to a 3-part-version driver, for example, a driver that takes the format of `xxx.yy.zz`, you will receive a `Failed to detect NVIDIA driver version.` message. This is due to a known bug in the entry point script's parsing of 3-part driver versions. This message is non-fatal and can be ignored. This will be fixed in the 19.04 release.

Chapter 96. PyTorch Release 18.12

The NVIDIA container image for PyTorch, release 18.12, is available.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in the `pytorch-py3.6` Conda™ environment in the container image.

The container also includes the following:

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 10.0.130 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 10.0.130
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.4.1
- NCCL 2.3.7 (optimized for NVLink™)
- APEx
- OpenMPI 3.1.2
- Caffe2
- TensorRT 5.0.2
- DALI 0.5.0 Beta
- Tensor Core Optimized Examples:
  - ResNet50 v1.5
  - GNMT v2

Driver Requirements

Release 18.12 is based on CUDA 10, which requires NVIDIA Driver release 410.xx. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 18.12 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 18.12 is based on PyTorch v0.4.1+ with up-to-date features from the PyTorch v1.0 preview (main branch up to PR 12303). PyTorch 0.4.1+ is released and included with this container.
- Performance improvement for PyTorch's native batch normalization.
- Mixed precision SoftMax enabling FP16 inputs, FP32 computations and FP32 outputs.
- Latest version of DALI 0.5.0 Beta.
- Ubuntu 16.04 with November 2018 updates

Tensor Core Examples

- An implementation of ResNet50. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model.
- An implementation of GNMT v2. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper.

Known Issues

Persistent batch normalization kernels have been disabled due to a known bug during validation. Batch normalization provides correct results and work as expected from users, however, this may cause up to 10% regression in time to solution performance on networks using batch normalization.

Chapter 97. PyTorch Release 18.11

The NVIDIA container image for PyTorch, release 18.11, is available.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in the `pytorch-py3.6` Conda™ environment in the container image.

The container also includes the following:

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 10.0.130 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 10.0.130
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.4.1
- NCCL 2.3.7 (optimized for NVLink™)
- APEx
- OpenMPI 3.1.2
- Caffe2
- TensorRT 5.0.2
- DALI 0.4.1 Beta
- Tensor Core Optimized Examples:
  - ResNet50 v1.5
  - GNMT v2

Driver Requirements

Release 18.11 is based on CUDA 10, which requires NVIDIA Driver release 410.xx. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 18.11 is based on PyTorch v0.4.1+ with up-to-date features from the PyTorch v1.0 preview (main branch up to PR 11834). PyTorch 0.4.1+ is released and included with this container.
- Latest version of NCCL 2.3.7.
- Latest version of NVIDIA cuDNN 7.4.1.
- Latest version of TensorRT 5.0.2
- Latest version of DALI 0.4.1 Beta.
- Ubuntu 16.04 with October 2018 updates

Tensor Core Examples

- An implementation of ResNet50. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model.
- An implementation of GNMT v2. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper.

Known Issues

There is a known bug when using persistent batch normalization kernels. If you are experiencing a drop in predictive power during testing and validation, the recommended workaround is to not add the `.eval()` flag on your model when doing testing or validation.

Chapter 98. PyTorch Release 18.10

The NVIDIA container image of PyTorch, release 18.10, is available.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in the `pytorch-py3.6` Conda™ environment in the container image.

The container also includes the following:

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 10.0.130 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 10.0.130
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.4.0
- NCCL 2.3.6 (optimized for NVLink™)
- Caffe2
- APEx
- OpenMPI 3.1.2
- TensorRT 5.0.0 RC
- DALI 0.4 Beta
- Tensor Core Optimized Examples:
  - ResNet50 v1.5
  - GNMT v2

Driver Requirements

Release 18.10 is based on CUDA 10, which requires NVIDIA Driver release 410.xx. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 18.10 is based on PyTorch v0.4.1+ with up-to-date features from the PyTorch v1.0 preview (main branch up to PR 11834). PyTorch 0.4.1+ is released and included with this container.
- When possible PyTorch will now automatically use cuDNN persistent RNN's providing improved speed for smaller RNN's.
- Improved multi-GPU performance in both PyTorch c10d and Apex's DDP.
- Faster weight norm with improved mixed-precision accuracy used through `torch.nn.utils.weight_norm`.
- Improved functionality of the `torch.jit.script` and `torch.jit.trace`preview features including better support for pointwise operations in fusion.
- Added support for a C++ only API (new PyTorch 1.0 preview feature).
- Dataloader may still throw a benign error when stopping iterations early, however, it is no longer preventing the process from ending.
- Latest version of DALI 0.4 Beta.
- Latest version of NCCL 2.3.6.
- Added support for OpenMPI 3.1.2
- Ubuntu 16.04 with September 2018 updates

Tensor Core Examples

- An implementation of ResNet50. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model.
- An implementation of GNMT v2. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper.

Known Issues

There are no new issues in this release.

Chapter 99. PyTorch Release 18.09

The NVIDIA container image of PyTorch, release 18.09, is available.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in the `pytorch-py3.6` Conda™ environment in the container image.

The container also includes the following:

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 10.0.130 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 10.0.130
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.3.0
- NCCL 2.3.4 (optimized for NVLink™)
- Caffe2
- TensorRT 5.0.0 RC
- DALI 0.2 Beta
- Tensor Core Optimized Examples:
  - ResNet50 v1.5
  - GNMT v2

Driver Requirements

Release 18.09 is based on CUDA 10, which requires NVIDIA Driver release 410.xx. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 18.09 is based on PyTorch 0.4.1+. PyTorch 0.4.1 is released and included with this container.
- Latest version of cuDNN 7.3.0.
- Latest version of CUDA 10.0.130 which includes support for DGX-2, Turing, and Jetson Xavier.
- Latest version of cuBLAS 10.0.130.
- Latest version of NCCL 2.3.4.
- Latest version of TensorRT 5.0.0 RC.

Note: All 18.09 containers inherit TensorRT 5.0.0 RC from the base container, however, some containers may not use TensorRT if there is no support for TensorRT in the given framework.

- An implementation of ResNet50. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model.
- Stream pool: PyTorch now uses per GPU stream pools behind the scenes. This means that CUDA streams are created when first used on a GPU and destroyed on exit. As a result, networks that use multiple streams may see the same stream used repeatedly in their profiles, and networks that retain streams for long periods may accidentally schedule parallelizable work to the same stream. It's recommended that streams be acquired, used, and released as needed.
- Reliability: Some cases where a dataloader could hang if shutdown during its iteration has been fixed.
- Fusion: Tensor and constant scalar operations, like `add(t, 1)`, and `chunk` operations are now fusable.
- Performance improvements: `dropout`, 1x1 convolutions for `NCHW`, and `weightnorm` should be faster in a majority of scenarios.
- Latest version of DALI 0.2 Beta
- Ubuntu 16.04 with August 2018 updates

Tensor Core Examples

- An implementation of ResNet50. The ResNet50 v1.5 model is a modified version of the original ResNet50 v1 model.
- An implementation of GNMT v2. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper.

Known Issues

- The DALI integrated ResNet-50 samples in the 18.09 NGC TensorFlow and PyTorch containers may result in lower than expected performance results. We are working to address the issue in the next release.
- There is a chance that PyTorch will hang on exit when running multi-gpu training. This hang does not affect any results of the run; however, the process will have to be terminated manually.

Chapter 100PyTorch Release 18.08

The NVIDIA container image of PyTorch, release 18.08, is available.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in the `pytorch-py3.6` Conda™ environment in the container image.

The container also includes the following:

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 9.0.176 (see Errata section and 2.1) including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.425
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.2.1
- NCCL 2.2.13 (optimized for NVLink™)
- Caffe2 0.8.1
- DALI 0.1.2 Beta
- Tensor Core Optimized Examples:
  - GNMT v2

Driver Requirements

Release 18.08 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 18.08 is based on PyTorch 0.4.1. PyTorch 0.4.1 is released and included with this container. See the release notes at https://github.com/pytorch/pytorch/releases for significant changes from PyTorch 0.4.
- Apex is now entirely Python for improved compatibility. Previous versions of Apex will not work with PyTorch 0.4.1 or newer versions.

- New ops in 18.08: `torch.pinverse`, `torch.unique`, `torch.erfc`, `torch.isinf`, `torch.isfinite`, `torch.reshape_as`.
- Support for cross-device gradient clipping.
- `torch.svd` and `torch.eig` in CUDA have been fixed, previously they could return incorrect results.
- An implementation of GNMT v2. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper.
- Latest version of cuDNN 7.2.1.
- Latest version of DALI 0.1.2 Beta.
- Added support for the Tensor Core Optimized Example: PyTorch GNMT model
- Ubuntu 16.04 with July 2018 updates

Usage

The `PYTHONPATH` environment variable in this container version has been updated to include all packages installed in the Conda environment and all PyTorch related packages. Users that rely on `PYTHONPATH` to point to local modules are advised to carefully check and set their `PYTHONPATH` variable in this container and moving forward.

Tensor Core Examples

An implementation of GNMT v2. The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper.

Known Issues

The DALI integrated ResNet-50 samples in the 18.08 NGC TensorFlow and PyTorch containers may result in lower than expected performance results. We are working to address the issue in the next release.

Chapter 101PyTorch Release 18.07

The NVIDIA container image of PyTorch, release 18.07, is available.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in the `pytorch-py3.6` Conda™ environment in the container image.

The container also includes the following:

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 9.0.176 (see Errata section and 2.1) including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.425
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.1.4
- NCCL 2.2.13 (optimized for NVLink™)
- Caffe2 0.8.1
- DALI 0.1 Beta

Driver Requirements

Release 18.07 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 18.07 is based on PyTorch 0.4.0 upstream main branch post commit cca2476.
- Clip grads can be used on a single tensor directly.
- The precision of MSELoss with half inputs has been improved.
- PyTorch's JIT (still in Alpha) now supports FP16 inputs and outputs, comparisons, the exp operator, and ReLU gates.
- Added support for DALI 0.1 Beta.
- Latest version of CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.425.

- Ubuntu 16.04 with June 2018 updates

Known Issues

When importing Caffe2 after importing Torch, there is an issue which causes GPU support for Caffe2 to be disabled. For users affected by this bug, it is recommended to either use the PyTorch 18.06 or 18.08 container.

Chapter 102PyTorch Release 18.06

The NVIDIA container image of PyTorch, release 18.06, is available.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in the `pytorch-py3.6` Conda™ environment in the container image.

The container also includes the following:

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 9.0.176 (see Errata section and 2.1) including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.333 (see section 2.3.1)
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.1.4
- NCCL 2.2.13 (optimized for NVLink™)
- Caffe2 0.8.1

Driver Requirements

Release 18.06 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 18.06 is based on PyTorch 0.4.0 upstream main branch post commit 0e9613c.
- Improved data loader pipeline in the ImageNet example, see `/opt/pytorch/examples/imagenet` within the container.
- Data loader pipeline now uses `pillow-simd` and `jpeg-turbo`.
- Improved FP16 support, specifically, reductions like `sum()` are now more accurate when using FP16.
- Improved distributed performance, specifically, gradient communication can now overlap with gradient computation in `backwards()`.
- Compatibility changes, specifically, Magma 1 is no longer supported.

- Ubuntu 16.04 with May 2018 updates

Known Issues

There are no known issues in this release.

Chapter 103PyTorch Release 18.05

The NVIDIA container image of PyTorch, release 18.05, is available.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in the `pytorch-py3.6` Conda™ environment in the container image.

The container also includes the following:

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 9.0.176 (see Errata section and 2.1) including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.333 (see section 2.3.1)
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.1.2
- NCCL 2.1.15 (optimized for NVLink™)
- Caffe2 0.8.1

Driver Requirements

Release 18.05 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 18.05 is based on PyTorch 0.4.0.
- Includes Caffe2 0.8.1. For more information, see PyTorch and Caffe2 repos getting closer together.
- APEx, an extension providing utilities for FP16 and multi-gpu training. For more information, see APEx: A PyTorch Extension and APEx.
- Ubuntu 16.04 with April 2018 updates

Known Issues

- Some mixed-precision models might encounter a crash due to a new FP16 overflow check added in PyTorch. We have an upstream fix submitted with PR 7382 and should be resolved in a future container.
- There is a minor performance regression with the imagenet sample in `/opt/pytorch/examples/imagnet` for some network architectures on multi-gpu cases. This regression will be fixed in the next release.

Chapter 104PyTorch Release 18.04

The NVIDIA container image of PyTorch, release 18.04, is available.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in the `pytorch-py3.6` Conda™ environment in the container image.

The container also includes the following:

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 9.0.176 (see Errata section and 2.1) including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.333 (see section 2.3.1)
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.1.1
- NCCL 2.1.15 (optimized for NVLink™)

Driver Requirements

Release 18.04 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 18.04 is based on PyTorch 0.3.1.
- Incorporated all upstream changes from the PyTorch main branch, specifically up to and including commit 2f27c1b5.
- Latest version of NCCL 2.1.15
- Ubuntu 16.04 with March 2018 updates

Known Issues

Some mixed-precision models might encounter a crash due to a new FP16 overflow check added in PyTorch. We have an upstream fix submitted with PR 7382 and should be resolved in a future container.

Chapter 105PyTorch Release 18.03

The NVIDIA container image of PyTorch, release 18.03, is available.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in the `pytorch-py3.6` Conda™ environment in the container image.

The container also includes the following:

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 9.0.176 (see Errata section and 2.1) including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.333 (see section 2.3.1)
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.1.1
- NCCL 2.1.2 (optimized for NVLink™)

Driver Requirements

Release 18.03 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- PyTorch container image version 18.03 is based on PyTorch 0.3.0.
- Incorporated all upstream changes from the PyTorch main branch, specifically, PR 5327.
- Latest version of cuBLAS 9.0.333
- Latest version of cuDNN 7.1.1
- Ubuntu 16.04 with February 2018 updates

Known Issues

There are no known issues in this release.

Chapter 106PyTorch Release 18.02

The NVIDIA container image of PyTorch, release 18.02, is available.

PyTorch container image version 18.02 is based on PyTorch 0.3.0.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in the `pytorch-py3.6` Conda™ environment in the container image.

The container also includes the following:

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 9.0.176 including:
  - CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.282 Patch 2 which is installed by default
  - cuBLAS 9.0.234 Patch 1 as a debian file. Installing Patch 1 by issuing the `dpkg -i /opt/cuda-cublas-9-0_9.0.234-1_amd64.deb` command is the workaround for the known issue described below.
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.0.5
- NCCL 2.1.2 (optimized for NVLink™)

Driver Requirements

Release 18.02 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- Improved multi-GPU performance on image networks shown in `/opt/pytorch/examples/imagenet`. You can run this example for multi-GPU by issuing the `python -m multiproc main.py` command.
- Latest version of cuBLAS
- Ubuntu 16.04 with January 2018 updates

Known Issues

cuBLAS 9.0.282 regresses RNN seq2seq FP16 performance for a small subset of input sizes. This issue should be fixed in the next update. As a workaround, install cuBLAS 9.0.234 Patch 1 by issuing the `dpkg -i /opt/cuda-cublas-9-0_9.0.234-1_amd64.deb` command.

Chapter 107PyTorch Release 18.01

The NVIDIA container image of PyTorch, release 18.01, is available.

PyTorch container image version 18.01 is based on PyTorch 0.3.0.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in the `pytorch-py3.6` Conda™ environment in the container image.

The container also includes the following:

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 9.0.176 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.282
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.0.5
- NCCL 2.1.2 (optimized for NVLink™)

Driver Requirements

Release 18.01 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- Latest version of cuBLAS
- Latest version of cuDNN
- Latest version of NCCL
- Ubuntu 16.04 with December 2017 updates

Known Issues

cuBLAS 9.0.282 regresses RNN seq2seq FP16 performance for a small subset of input sizes. As a workaround, revert back to the 11.12 container.

Chapter 108PyTorch Release 17.12

The NVIDIA container image of PyTorch, release 17.12, is available.

PyTorch container image version 17.12 is based on PyTorch 0.2.0.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in the `pytorch-py35` Conda™ environment in the container image.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA 9.0.176 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.234
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.0.5
- NCCL 2.1.2 (optimized for NVLink™)

Driver Requirements

Release 17.12 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- Latest version of CUDA
- Latest version of cuDNN
- Latest version of NCCL
- Ubuntu 16.04 with November 2017 updates

Known Issues

There are no known issues in this release.

Chapter 109PyTorch Release 17.11

The NVIDIA container image of PyTorch, release 17.11, is available.

PyTorch container image version 17.11 is based on PyTorch 0.2.0.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in the `pytorch-py35` Conda™ environment in the container image.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA 9.0.176 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.234
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.0.4
- NCCL 2.1.2 (optimized for NVLink™)

Driver Requirements

Release 17.11 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- Tensor Op accelerated RNNs for Volta architecture
- Improved depthwise separable convolution performance
- Improved automatic differentiation engine latency
- Latest version of CUDA
- Latest version of cuDNN
- Latest version of NCCL
- Ubuntu 16.04 with October 2017 updates

Known Issues

There are no known issues in this release.

Chapter 110PyTorch Release 17.10

The NVIDIA container image of PyTorch, release 17.10, is available.

PyTorch container image version 17.10 is based on PyTorch 0.2.0.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in the `pytorch-py35` Conda™ environment in the container image.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA® 9.0
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.0.3
- NVIDIA® Collective Communications Library™ (NCCL) 2.0.5 (optimized for NVLink™)

Driver Requirements

Release 17.10 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- Latest version of CUDA
- Latest version of cuDNN
- Latest version of NCCL
- Ubuntu 16.04 with September 2017 updates

Known Issues

There are no known issues in this release.

Chapter 111PyTorch Release 17.09

The NVIDIA container image of PyTorch, release 17.09, is available.

PyTorch container image version 17.09 is based on PyTorch 0.2.0.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed in the `pytorch-py35` Conda™ environment in the container image.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA® 9.0
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.0.2
- NVIDIA® Collective Communications Library ™ (NCCL) 2.0.5 (optimized for NVLink™)

Driver Requirements

Release 17.09 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- Supports Tensor Core operations for convolutions and GEMMs on Volta hardware
- The `examples` directory contains examples of ImageNet and LSTM training scripts that use FP16 data, as well as show how to train with FP16
- Matrix multiplication on FP16 inputs uses Tensor Core math when available
- A custom batch normalization layer is implemented to use cuDNN for batch normalization with FP16 inputs
- Latest version of CUDA
- Latest version of cuDNN with support for Tensor Core math when available
- Latest version of NCCL
- Ubuntu 16.04 with August 2017 updates

Known Issues

There are no known issues in this release.

Chapter 112PyTorch Release 17.07

The NVIDIA container image of PyTorch, release 17.07, is available.

PyTorch container image version 17.07 is based on PyTorch 0.1.12.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed into the `/usr/local/[bin,share,lib]` directories in the container image.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA® 8.0.61.2 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) Patch 2
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 6.0.21
- NVIDIA® Collective Communications Library ™ (NCCL) 2.0.3 (optimized for NVLink™)

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- Support for advanced tensor indexing
- Support multi-node or multi-process mode on the same node
- Support for double backward for most functions, including convolution
- Ubuntu 16.04 with June 2017 updates

Known Issues

There are no known issues in this release.

Chapter 113PyTorch Release 17.06

The NVIDIA container image of PyTorch, release 17.06, is available.

PyTorch container image version 17.06 is based on PyTorch 0.1.12.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed into the `/usr/local/[bin,share,lib]` directories in the container image.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA® 8.0.61
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 6.0.21
- NVIDIA® Collective Communications Library ™ (NCCL) 1.6.1 (optimized for NVLink™)

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- Ubuntu 16.04 with May 2017 updates

Known Issues

The NCCL library version 1.6.1 included in this image, modifies the output buffers on all GPUs during in-place `ncclReduce()` operations, whereas normally only the "root" (target) device's output buffer should be modified. This is fixed in later versions of NCCL, as will be packaged in later versions of this image. As a workaround, either use `ncclAllReduce()`, which correctly modifies output buffers of all GPUs to the same values, or use out-of-place `ncclReduce()`, wherein the output buffer is distinct from the input buffer.

Chapter 114PyTorch Release 17.05

The NVIDIA container image of PyTorch, release 17.05, is available.

PyTorch container image version 17.05 is based on PyTorch 0.1.12.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed into the `/usr/local/[bin,share,lib]` directories in the container image.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA® 8.0.61
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 6.0.21
- NVIDIA® Collective Communications Library ™ (NCCL) 1.6.1 (optimized for NVLink™)

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- Latest cuDNN release
- Ubuntu 16.04 with April 2017 updates

Known Issues

The NCCL library version 1.6.1 included in this image, modifies the output buffers on all GPUs during in-place `ncclReduce()` operations, whereas normally only the "root" (target) device's output buffer should be modified. This is fixed in later versions of NCCL, as will be packaged in later versions of this image. As a workaround, either use `ncclAllReduce()`, which correctly modifies output buffers of all GPUs to the same values, or use out-of-place `ncclReduce()`, wherein the output buffer is distinct from the input buffer.

Chapter 115PyTorch Release 17.04

The NVIDIA container image of PyTorch, release 17.04, is available.

PyTorch container image version 17.04 is based on PyTorch 0.1.10.

Contents of PyTorch

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is pre-built and installed into the `/usr/local/[bin,share,lib]` directories in the container image.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA® 8.0.61
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 6.0.20
- NVIDIA® Collective Communications Library ™ (NCCL) 1.6.1 (optimized for NVLink™)

Key Features and Enhancements

This PyTorch release includes the following key features and enhancements.

- Reduce DataParallel overhead on more than 4 GPUs
- cuDNN v6 integration
- Synced to upstream PyTorch version as of March 2017
- Ubuntu 16.04 with March 2017 updates

Known Issues

There are no known issues in this release.

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
https://www.nvidia.com

Trademarks

NVIDIA, the NVIDIA logo, and cuBLAS, CUDA, DALI, DGX, DGX-1, DGX-2, DGX Station, DLProf, Jetson, Kepler, Maxwell, NCCL, Nsight Compute, Nsight Systems, NvCaffe, PerfWorks, Pascal, SDK Manager, Tegra, TensorRT, Triton Inference Server, Tesla, TF-TRT, and Volta are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2017-2026 NVIDIA Corporation & Affiliates. All rights reserved.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
https://www.nvidia.com