NVIDIA Secure AI

Operations Guide

DU-12609-001_v01 | October 2025

Document History

DU-12609-001_v01

Version	Date	Authors	Description of Change
01	November 2025	NVIDIA Corporation	Initial Release

Table of Contents

Overview

The *NVIDIA SecureAI Operations Guide* provides instructions, recommendations, and cautions for the initial deployment of an NVIDIA SecureAI solution. It also provides considerations for developing applications for an NVIDIA SecureAI solution.

Deployment Tips - Dos and Don’ts

Ensure that your Hardware and Software Are Compatible with NVIDIA Secure AI

The functionality of NVIDIA Secure AI solutions and their support depend on the hardware platform, driver, and firmware versions. To ensure compatibility, check your infrastructure against the Secure AI Compatibility Matrix.

Ensure that Persistence Mode is On

In a typical operation, when the NVIDIA device resources are no longer being used, the NVIDIA kernel driver tears down the device state. However, in CC mode, this behavior causes the shared-secret and the shared keys that were established during the setup SPDM phase of the driver to be destroyed. To protect user data, the GPU does not allow the establishment of an SPDM session to be restarted without an FLR, which resets and scrubs the GPU.

The `nvidia-persistenced` daemon provides the persistence mode configuration option, which can be set by NVIDIA management software, such as nvidia-smi. When persistence mode is on, the NVIDIA kernel driver is prevented from exiting. The `nvidia-persistenced` daemon does not use any device resources. It merely sleeps while maintaining a reference to the NVIDIA device state.

Do Not Disable the GPU Ready-State

The GPU ready-state is set by the driver to track whether the infrastructure has passed attestation requirements. If the ready-state is set to 0 after attestation, the driver fails to load. The only way to recover from this condition is to reset the entire virtual machine.

Do Not Install the GPU Driver on the Host

Typically, the GPU is bound to a stub driver, which hides it from the host OS and makes it available for pass through. If you install the GPU driver on the host and the host binds to that GPU, the GPU will not be available for pass through to the guest, because the host driver controls the GPU.

Ensure that the Host Has Twice as Much Memory as Can Be Allocated to VMs

An issue with QEMU in versions 10.1 and lower requires the host to have twice as much memory as can be allocated to VMs. QEMU 10.1 has a fix for the issue, but this fix is natively available only starting with Ubuntu 25.10.

Use OpenSSL 3.5.0 or Newer

OpenSSL 3.5.0 and newer versions take advantage of e AVX512 instructions to deliver better encryption performance. CC architectures based on bounce buffers benefit most from this increase in performance.

Application Development Considerations

Some CUDA API features are not supported by NVIDIA Secure AI.

Some limitations arise because Confidential Computing is inherently incompatible with certain CUDA constructs. For example:

- Memory handles cannot be propagated beyond the scope of a Trusted Compute Boundary (TCB).
- Developer tools such as Nsight systems are blocked by firewalls in CC mode.

Some limitations are implementation specific. For example, in SPT CC Mode, no more than one GPU can be passed through to a CVM.

The following sections summarize support for Secure AI modes.

Peer Memory Access

API	SPT	PPCIe
Runtime API: - `cudaDeviceCanAccessPeer` - `cudaDeviceDisablePeerAccess` - `cudaDeviceEnablePeerAccess`	Returns `CUDA_ERROR_NOT_SUPPORTED`	Supported
Driver API: - `cuDeviceCanAccessPeer` - `cuCtxDisablePeerAccess` - `cuCtxEnablePeerAccess` - `cuDeviceGetP2PAttribute`	Returns `CUDA_ERROR_NOT_SUPPORTED`	Supported

Memory Management

API	SPT	PPCIe
Runtime API: > cudaMemcpyAsync > cudaMemcpy2D > cudaMemcpy2DAsync > cudaMemcpy3D > cudaMemcpy3DAsync > cudaMemcpy3DBatchAsync > cudaMemcpyBatchAsync > cudaMemcpyFromSymbol > cudaMemcpyFromSymbolAsync > cudaMemcpyToSymbol > cudaMemcpyToSymbolAsync	Non-block-linear memcopy supported.	Non-block-linear memcopy supported. Peer-to-peer copies using CUDA APIs (for example, cudaMemcpyAsync) are supported if peer access is successfully enabled between the source and destination devices using cudaDeviceEnablePeerAccess or cuCtxEnablePeerAccess. Peer-to-Host-to-Peer copies are not supported and APIs falling back to these shall fail..
Driver API: > cuMemcpyAsync > cuMemcpy2D > cuMemcpy2DAsync > cuMemcpy3D > cuMemcpy3DAsync > cuMemcpy3DBatchAsync > cuMemcpyBatchAsync > cuMemcpy2DUnaligned > cuMemcpyDtoD > cuMemcpyDtoDAsync > cuMemcpyDtoH > cuMemcpyDtoHAsync > cuMemcpyHtoD > cuMemcpyHtoDAsync	Non-block-linear memcopy supported	Non-block-linear memcopy supported. Peer-to-peer copies using CUDA APIs (for example, cudaMemcpyAsync) are supported if peer access is successfully enabled between the source and destination devices using cudaDeviceEnablePeerAccess or cuCtxEnablePeerAccess. Peer-to-Host-to-Peer copies are not supported and APIs falling back to these shall fail.
Runtime API: > cudaMemcpy3DPeer > cudaMemcpy3DPeerAsync > cudaMemcpyPeer > cudaMemcpyPeerAsync	Returns CUDA_ERROR_NOT_SUPPORTED	Returns CUDA_ERROR_NOT_SUPPORTED
Driver API: > cuMemBatchDecompressAsync > cuMemcpy3DPeer > cuMemcpy3DPeerAsync > cuMemcpyPeer > cuMemcpyPeerAsync	Returns CUDA_ERROR_NOT_SUPPORTED	Returns CUDA_ERROR_NOT_SUPPORTED

Host Memory Registration

API	SPT	PPCIe
Runtime API: > cudaHostRegister > cudaHostUnregister	Returns CUDA_ERROR_NOT_SUPPORTED	Returns CUDA_ERROR_NOT_SUPPORTED
Driver API: > cuMemHostRegister > cuMemHostUnregister	Returns CUDA_ERROR_NOT_SUPPORTED	Returns CUDA_ERROR_NOT_SUPPORTED

CUDA Array Copies (Block Linear)

API	SPT	PPCIe
Runtime API: > cudaMemcpy2DFromArray > cudaMemcpy2DFromArrayAsync > cudaMemcpy2DToArray > cudaMemcpy2DToArrayAsync > cudaMemcpy3D > cudaMemcpy3DAsync > cudaMemcpy3DPeer > cudaMemcpy3DPeerAsync	Returns CUDA_ERROR_NOT_SUPPORTED	Returns CUDA_ERROR_NOT_SUPPORTED
Driver API: > cuMemcpy2D > cuMemcpy2DAsync > cuMemcpy2DUnaligned > cuMemcpy3D > cuMemcpy3DAsync > cuMemcpyAtoA > cuMemcpyAtoH > cuMemcpyAtoHAsync > cuMemcpyHtoA > cuMemcpyHtoAAsync	Returns CUDA_ERROR_NOT_SUPPORTED	Returns CUDA_ERROR_NOT_SUPPORTED

Stream Memory Operations

API	SPT	PPCIe
Driver API: > cuStreamBatchMemOp > cuStreamBatchMemOp_v2 > cuStreamWaitValue32 > cuStreamWaitValue32_v2 > cuStreamWaitValue64 > cuStreamWaitValue64_v2 > cuStreamWriteValue32 > cuStreamWriteValue32_v2 > cuStreamWriteValue64 > cuStreamWriteValue64_v2 > cuGraphAddBatchMemOpNode > cuGraphBatchMemOpNodeGetParams > cuGraphBatchMemOpNodeSetParams > cuGraphExecBatchMemOpNodeSetParams	Passing pointers allocated using the cudaMallocHost, cudaHostAlloc, or cuMemAllocHost API to stream memory operation APIs returns CUDA_ERROR_NOT_SUPPORTED.	

IPC

API	SPT	PPCIe
Runtime API: > cudaIpcGetEventHandle > cudaIpcOpenEventHandle > cudaIpcGetMemHandle > cudaIpcOpenMemHandle > cudaIpcCloseMemHandle	Supported	Returns CUDA_ERROR_NOT_SUPPORTED
Driver API: > cuIpcGetEventHandle > cuIpcOpenEventHandle > cuIpcGetMemHandle > cuIpcOpenMemHandle > cuIpcCloseMemHandle	Supported	Returns CUDA_ERROR_NOT_SUPPORTED

CUDA Events

API	SPT	PPCIe
Runtime API: > cudaEventElapsedTime	Inaccurate. Not recommended	Inaccurate. Not recommended

API	SPT	PPCIe
Driver API: > cuEventElapsedTime	Inaccurate. Not recommended	Inaccurate. Not recommended

External Memory and Semaphore Interoperability

API	SPT	PPCIe
Runtime API: > cudaExternalMemoryGetMappedBuffer > cudaExternalMemoryGetMappedMipmappedArray > cudaDestroyExternalMemory > cudaFreeMipmappedArray > cudaImportExternalSemaphore > cudaSignalExternalSemaphoresAsync > cudaWaitExternalSemaphoresAsync > cudaDestroyExternalSemaphore > cudaGraphAddExternalSemaphoresSignalNode > cudaGraphAddExternalSemaphoresWaitNode > cudaGraphExecExternalSemaphoresSignalNodeSetParams > cudaGraphExecExternalSemaphoresWaitNodeSetParams > cudaGraphExternalSemaphoresSignalNodeGetParams > cudaGraphExternalSemaphoresSignalNodeSetParams > cudaGraphExternalSemaphoresWaitNodeGetParams > cudaGraphExternalSemaphoresWaitNodeSetParams	Returns CUDA_ERROR_NOT_SUPPORTED	Returns CUDA_ERROR_NOT_SUPPORTED
Driver API: > cuImportExternalMemory > cuExternalMemoryGetMappedBuffer > cuExternalMemoryGetMappedMipmappedArray > cuDestroyExternalMemory > cuFreeMipmappedArray > cuImportExternalSemaphore > cuSignalExternalSemaphoresAsync > cuWaitExternalSemaphoresAsync > cuDestroyExternalSemaphore > cuGraphAddExternalSemaphoresSignalNode > cuGraphAddExternalSemaphoresWaitNode > cuGraphExecExternalSemaphoresSignalNodeSetParams > cuGraphExecExternalSemaphoresWaitNodeSetParams > cuGraphExternalSemaphoresSignalNodeGetParams > cuGraphExternalSemaphoresSignalNodeSetParams > cuGraphExternalSemaphoresWaitNodeGetParams > cuGraphExternalSemaphoresWaitNodeSetParams	Returns CUDA_ERROR_NOT_SUPPORTED	Returns CUDA_ERROR_NOT_SUPPORTED

OpenGL Interoperability

API	SPT	PPCIe
Runtime API: > cudaGLGetDevices > cudaGraphicsGLRegisterBuffer > cudaGraphicsGLRegisterImage > cudaWGLGetDevice	Returns CUDA_ERROR_NOT_SUPPORTED	Returns CUDA_ERROR_NOT_SUPPORTED
Driver API: > cuGLGetDevices > cuGraphicsGLRegisterBuffer > cuGraphicsGLRegisterImage > cuWGLGetDevice	Returns CUDA_ERROR_NOT_SUPPORTED	Returns CUDA_ERROR_NOT_SUPPORTED

Direct3D Interoperability

API	SPT	PPCIe
Runtime API: > cudaD3D9GetDevice > cudaD3D9GetDevices > cudaD3D9GetDirect3DDevice > cudaD3D9SetDirect3DDevice > cudaGraphicsD3D9RegisterResource > cudaD3D10GetDevice > cudaD3D10GetDevices > cudaGraphicsD3D10RegisterResource > cudaD3D11GetDevice > cudaD3D11GetDevices > cudaGraphicsD3D11RegisterResource	Returns CUDA_ERROR_NOT_SUPPORTED	Returns CUDA_ERROR_NOT_SUPPORTED
Driver API: > cuD3D9CtxCreate > cuD3D9CtxCreateOnDevice > cuD3D9GetDevice > cuD3D9GetDevices > cuD3D9GetDirect3DDevice > cuGraphicsD3D9RegisterResource > cuD3D10GetDevice > cuD3D10GetDevices > cuGraphicsD3D10RegisterResource > cuD3D11GetDevice > cuD3D11GetDevices > cuGraphicsD3D11RegisterResource	Returns CUDA_ERROR_NOT_SUPPORTED	Returns CUDA_ERROR_NOT_SUPPORTED

VDPAU Interoperability

API	SPT	PPCIe
Runtime API: > cudaGraphicsVDPAURegisterOutputSurface > cudaGraphicsVDPAURegisterVideoSurface > cudaVDPAUGetDevice > cudaVDPAUSetVDPAUDevice	Returns CUDA_ERROR_NOT_SUPPORTED	Returns CUDA_ERROR_NOT_SUPPORTED
Driver API: > cuGraphicsVDPAURegisterOutputSurface > cuGraphicsVDPAURegisterVideoSurface > cuVDPAUCtxCreate > cuVDPAUGetDevice	Returns CUDA_ERROR_NOT_SUPPORTED	Returns CUDA_ERROR_NOT_SUPPORTED

EGL Interoperability

API	SPT	PPCIe
Runtime API: > cudaEGLStreamConsumerAcquireFrame > cudaEGLStreamConsumerConnect > cudaEGLStreamConsumerConnectWithFlags > cudaEGLStreamConsumerDisconnect > cudaEGLStreamConsumerReleaseFrame > cudaEGLStreamProducerConnect > cudaEGLStreamProducerDisconnect > cudaEGLStreamProducerPresentFrame > cudaEGLStreamProducerReturnFrame > cudaEventCreateFromEGLSync > cudaGraphicsEGLRegisterImage > cudaGraphicsResourceGetMappedEglFrame	Returns CUDA_ERROR_NOT_SUPPORTED	Returns CUDA_ERROR_NOT_SUPPORTED
Driver API: > cuEGLStreamConsumerAcquireFrame > cuEGLStreamConsumerConnect > cuEGLStreamConsumerConnectWithFlags > cuEGLStreamConsumerDisconnect > cuEGLStreamConsumerReleaseFrame > cuEGLStreamProducerConnect > cuEGLStreamProducerDisconnect > cuEGLStreamProducerPresentFrame > cuEGLStreamProducerReturnFrame > cuEventCreateFromEGLSync > cuGraphicsEGLRegisterImage > cuGraphicsResourceGetMappedEglFrame	Returns CUDA_ERROR_NOT_SUPPORTED	Returns CUDA_ERROR_NOT_SUPPORTED

Unified Addressing

API	PPCIe
Driver API: > cuPointerGetAttribute	Calling this API with pointers allocated by using `cudaMallocHost` returns one of the following values:: > `CU_MEMORYTYPE_DEVICE` for `CU_POINTER_ATTRIBUTE_MEMORY_TYPE` > 1 for `CU_POINTER_ATTRIBUTE_IS_MANAGED` > `CUDA_ERROR_INVALID_DEVICE` for `CU_POINTER_ATTRIBUTE_P2P_TOKENS` > 1 for `CU_POINTER_ATTRIBUTE_SYNC_MEMOPS`

Virtual Memory Management

API	SPT	PPCIe
Driver API: > cuMemImportFromShareableHandle	The behavior is identical to the behavior described in the public CUDA API documentation.	When the source or destination operand is imported, for GPU memory allocations on a device that is not visible to the importing process, then host-to-device, or device-to-host copies launched from the importing process and using the imported operands might fail asynchronously with `cudaErrorLaunchFailure`.

Multicast Object Management

API	SPT	PPCIe
Driver API: > `cuMulticastAddDevice` > `cuMulticastBindAddr` > `cuMulticastBindMem` > `cuMulticastCreate` > `cuMulticastGetGranularity` > `cuMulticastUnbind`	Returns `CUDA_ERROR_NOT_SUPPORTED`	Returns `CUDA_ERROR_NOT_SUPPORTED`

Programmatic Dependent Launch and Synchronization

The CUDA Programmatic Dependent Launch and Synchronization feature does not show expected overlaps in primary and secondary kernel execution. A program that uses these APIs should functionally succeed in CC modes.

Using Programmatic Dependent Launch with CUDA Graphs still works in the same way as in non-CC mode.

CUDA Context Management

The maximum of contexts possible depends on the system configuration and GPU attributes. The limitation on the maximum number of contexts arises from a system-wide restriction on the number of secure copy channels that the GPU DMA engines support.

Unsupported Features

NVIDIA Secure AI does **not** support the following features:

- CUDA Minor Version Compatibility
- CUDA Forward Compatibility
- GPUDirect RDMA
- CBL
- Multi-Process Service (MPS)
- Multi-Instance GPU (MIG)

Additional Resources

Visit the NVIDIA Trusted Computing Solutions page for additional documentation.

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.

Trademarks

NVIDIA, the NVIDIA logo 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.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com