# Physics Informed RNN-DCT Networks for Time-Dependent Partial Differential Equations

**Benjamin Wu**[*][†]
NVIDIA
National Astronomical Observatory of Japan
benwu.astro@gmail.com

**Oliver Hennigh**
NVIDIA
ohennigh@nvidia.com

**Jan Kautz**
NVIDIA
jkautz@nvidia.com

**Sanjay Choudhry**
NVIDIA
schoudhry@nvidia.com

**Wonmin Byeon**[*]
NVIDIA
wbyeon@nvidia.com

## Abstract

Physics-informed neural networks allow models to be trained by physical laws described by general nonlinear partial differential equations. However, traditional architectures struggle to solve more challenging time-dependent problems. In this work, we present a novel physics-informed framework for solving time-dependent partial differential equations. Our proposed model utilizes discrete cosine transforms to encode spatial frequencies and recurrent neural networks to process the time evolution, achieving state-of-the-art performance on the Taylor-Green vortex relative to other physics-informed baseline models.

## 1 Introduction

Numerical simulations have become an indispensable tool for modeling physical systems, which in turn drive advancements in engineering and scientific discovery. However, as the physical complexity or spatio-temporal resolution of a simulation increases, the computational resources and run times required to solve the governing partial differential equations (PDEs) often grow drastically.

Recently, machine learning approaches have been applied to the domain of physical simulation to ameliorate these issues by approximating traditional solvers with faster, less resource-intensive ones. These methods generally fall into two main paradigms: data-driven supervision [1–5] or physics-informed neural networks (PINNs) [6–9]. *PINN-based solvers* parameterize the solution function directly as a nueral network. This is typically done by passing a set of query points through a feed-forward fully-connected neural network (or multilayer perceptron, MLP) and minimizing a loss function based on the governing PDEs, initial conditions (ICs) and boundary conditions (BCs). This allows the simulation to be constrained by physics alone and does not require any training data. However, the accuracy of traditional PINN-based approaches is limited to problems in low dimensions and with simpler time-independent physics.

Although PINNs provide a well-principled, machine learning approach that promises to revolutionize numerical simulations, their current constraints to problems with simple geometries and short times severely limits their real-world impact. We address these shortcomings by introducing novel design choices that improve the simulation accuracy and efficiency of PINN solvers on more challenging problems, particularly in the regime of long time evolution where current PINNs severely struggle.

[*]Equal contribution.
[†]Work done during NVIDIA AI Research Residency

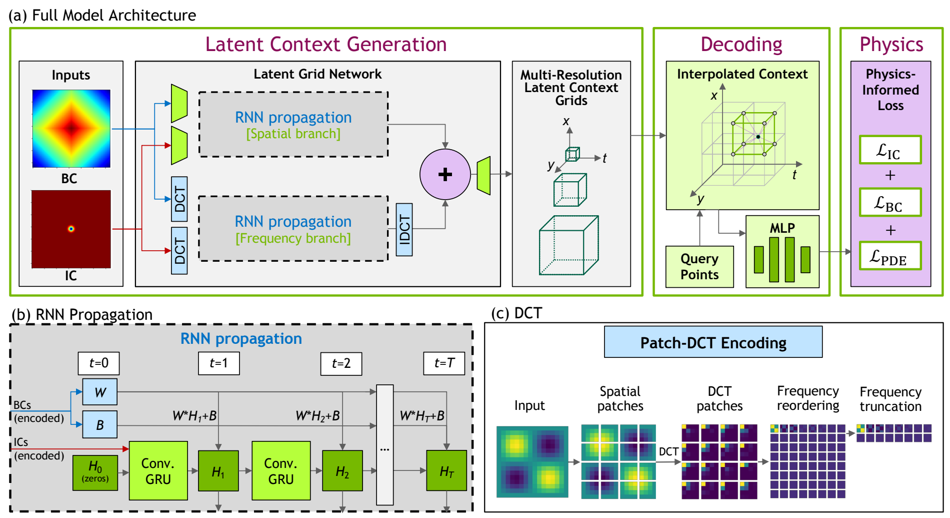

Figure 1: (a) Full model architecture. (b) RNN propagation. (c) DCT encoding.

Our key contributions are as follows: **(1)** We propose a new approach for generating a grid of latent context vectors to condition the spatio-temporal query points entering the MLP. Our method requires no additional data and enables PINNs to learn complex time-dependent physics problems. **(2)** Our work is the first to directly address space-time-dependent physics end-to-end in PINNs with RNNs. Unlike previous approaches, our model does not need a separate method to handle the time dimension. This is achieved by utilizing convolutional gated recurrent units (ConvGRUs) for learning the spatio-temporal dynamics of simulations. **(3)** We separate the spatial and frequency domains, adding flexibility for the network to learn more diverse physical problems. **(4)** We test the new model against other architectures on a benchmark transient simulation problem and demonstrate state-of-the-art results in accuracy.

## 2 Methods

In this paper, we propose a new model that enables PINN-based neural solvers to learn temporal dynamics in both the spatial and frequency domains. Using no additional data, our architecture can generate a latent context grid that efficiently represents more challenging spatio-temporal physical problems. Our full architecture is shown in Figure 1 (a). It consists of three primary parts, which are explained in more detail below: (1) latent context generation, (2) decoding, and (3) physics-informing.

**Latent grid network** The primary novelty of our method is the latent grid network that can generate context grids which efficiently represent the entire spatio-temporal domain of a physical problem without the need for additional data.

This network requires two inputs for the problem-specific constraints: ICs and BCs. The ICs are defined as $u_0 = u(x_{1,..,N}, t = 0)$ for each PDE solution function $u$ over $N$ spatial dimensions. The BCs are defined based on the geometry of the problem for each spatial dimension. An additional spatial weighting by signed distance functions (SDFs) can also be applied to avoid discontinuities at, e.g., physical boundaries, but would not be necessary for, e.g., periodic BCs. Each tensor undergoes an encoding step in either the frequency or spatial domain.

After compression, the representations enter the *RNN propagation stage* (Figure 1 (b)), in which the BCs are split into an additive ($B^{\text{bc}}$) and multiplicative ($W^{\text{bc}}$) component and combined with an IC-informed state matrix ($H_t$). The final output at each timestep is computed as $S_t = W^{\text{bc}}H_t + B^{\text{bc}}$. This method offers flexibility and efficiency in learning the dynamics of compressed simulations [3]. To predict the simulation state at each successive timestep, the previous hidden state $H_{t-1}$ is passed through a convolutional GRU (ConvGRU) along with the previous output $S_{t-1}$; for timestep 0, the initial state $H_0$ set to zero and ICs are used as inputs. This occurs in a recurrent manner until the final time $T$. Thus, for each timestep, the RNN propagation stage outputs $S_t$

which is then sent to a decoding step corresponding to the original frequency or spatial encoding: $S_0 = u_0,\ H_0 = \mathbf{0},\ H_t = \text{ConvGRU}(S_{t-1}, H_{t-1}),\ S_t = W^{\text{bc}} H_t + B^{\text{bc}},\ t \in \{1, \ldots, T\}$.

The RNN propagation stage is duplicated across two branches: frequency and spatial. The *frequency branch* transforms the spatial inputs to frequencies via the discrete cosine transform (DCT), motivated by [10]. Figure 1 (c) illustrates our patch-wise DCT encoding step. First, the ICs and BCs are separately split into spatial patches of size $p \times p$. DCTs are performed on each patch to yield the corresponding $p \times p$ frequency coefficient array. The tensor is then reshaped such that the same coefficient across all patches forms each channel, and the channels are reordered by increasing coefficient (i.e., decreasing energy). After the reordering, the channels are truncated by $n\%$, so the lowest $n\%$ of frequency coefficients (largest energies) are kept. This outputs highly compressed representations for the ICs and BCs, which are used as inputs for an RNN propagation branch that occurs completely in the frequency domain.

The *spatial branch* follows a traditional ResNet [11] architecture, in which the ICs and BCs each pass through separate convolutional encoders consisting of sets of convolutional blocks with residual connections. The inputs are downsampled with strided convolutions before entering the RNN propagation stage in the spatial domain.

After RNN propagation, the outputs are combined to form the latent grid. In the frequency branch, the output state at each timestep from the RNN is converted back into the spatial domain. This is done by reshaping the frequencies from coefficients to patches, performing IDCTs, and then merging the patches to reconstruct the spatial domain. The output of the frequency branch is denoted as $O_t^f$. The representation in the spatial domain $O_t^s$ is then added with learnable weights $W_t^o$. Thus, the final output is computed as: $O_t = W_t^o O_t^s + O_t^f$. These combined outputs $O_t$ for each timestep are used to form the spatio-temporal latent context grids. Finally, grids at multiple resolutions are generated by upsampling the outputs $O_t$ using transpose convolutional blocks.

**Decoding step** The multi-resolution latent context grids generated from the previous step are then used to query points input to the MLP. This decoding step follows the same principles as [12]. Given a random query point $\mathbf{x} := (x, y, t)$, $k$ neighboring vertices of the query point at each dimension are selected. Using these neighboring vertices, the final values of the context vector are then interpolated using Gaussian interpolation. This process is repeated for each of the multi-resolution grids allowing the PINN framework to learn multi-scale spatio-temporal quantities.

**Physics-informed loss** The MLP outputs predictions that are then subject to a loss function determined by the ICs, BCs, and the PDEs. The losses are backpropagated through the entire combined decoding and latent grid network and minimized via stochastic gradient descent. This end-to-end training allows our two-branch convGRU model to learn accurate time-evolution of the spatial and frequency domains in complex physical problems.

## 3 Experiments

We compare our model (**RNN-SpDCT**) against several other neural solver architectures using the time-dependent 2D Taylor-Green vortex problem. This problem is commonly used to test and validate spatial and temporal accuracy of both traditional and ML-based fluid solvers. We compare against other PINN-based models and use identical ICs, BCs, and PDE constraints for each. We used a single Tesla V100 16G or 32G for all experiments. The patch-wise DCTs use more GPU memory than other models. The implementation details are not described due to the page limit. We will include the details in the final version.

**Baseline models** We compare our proposed model against several PINN-based approaches: MLP-PINN, RNN-S, RNN-pDCT, and RNN-SfDCT. All comparing models contain the RNN-propagation and decoding steps except for MLP-PINN and all use a physics informed loss explained in section 2. All use the same hyper-parameters as our model except for learning rates and decay steps. **MLP-PINN**: a traditional MLP-based PINN solver used as a default model from `SimNet` [13]. **RNN-S**: a PINN solver with a latent grid network consisting of a single spatial branch (ResNet). **RNN-pDCT**: a PINN solver with a latent grid network consisting of a single frequency branch (DCT). **RNN-SfDCT**:a PINN solver with a latent grid network consisting of both spatial and frequency branches.

Table 1: Quantitative comparisons on the Taylor-Green vortex. The model is trained and tested for $2\pi$ seconds. $\nu$ is the kinematic viscosity of the fluid. F and S indicate frequency and spatial branches. FullDCT applies DCT to the entire input. All tabulated values have been multiplied by $10^2$ for readability.

| | | | Taylor-Green Vortex | | | | | |
|---|---|---|---|---|---|---|---|---|
| Model Name | Branch | DCT type | $\nu = 1.0$ | | $\nu = 0.1$ | | $\nu = 0.01$ | |
| | | | velocity | pressure | velocity | pressure | velocity | pressure |
| MLP-PINN | - | - | 0.033 | 5.910 | 1.769 | 0.782 | 0.824 | 0.522 |
| RNN-S | S | - | 6.683e-8 | 0.075 | 2.975e-7 | 0.138 | 2.527e-7 | 0.020 |
| RNN-pDCT | F | patch | 1.979e-6 | 0.172 | 5.957e-7 | 1.383 | 8.804e-7 | 0.508 |
| RNN-SfDCT | S+F | full | 9.171e-8 | 1.177 | 2.961e-7 | 0.301 | 7.015e-6 | 0.018 |
| **RNN-SpDCT** | S+F | patch | 1.408e-7 | **0.044** | 3.107e-7 | **0.101** | 1.328e-6 | **0.012** |

The frequency branch in this model applies DCT/IDCT to the full input, foregoing the patching, coefficient channel reordering, and truncation steps.

### 3.1 Taylor-Green vortex

The Taylor-Green vortex describes a decaying vortex flow which follows a special case of the Navier-Stokes equations [14]. The incompressible Navier-Stokes equations in 2D are:

$$\begin{aligned} \partial_x u + \partial_y v &= 0 \\ \partial_t u + u\partial_x u + v\partial_y u &= -\partial_x \rho/\rho + \nu(\partial_{xx} u + \partial_{yy} u) \\ \partial_t v + u\partial_x v + v\partial_y v &= -\partial_y \rho/\rho + \nu(\partial_{xx} v + \partial_{yy} v). \end{aligned} \tag{1}$$

where $u$ and $v$ are the $x$- and $y$-velocities, respectively, $\nu \in \mathbb{R}_+$ is the kinematic viscosity, and $\rho$ is the density.

The exact closed form solution for the Taylor-Green vortex over the domain $x \times y \times t \in [0, 2\pi] \times [0, 2\pi] \times [0, T]$ is:

$$\begin{aligned} u &= \cos x \sin y F(t) \\ v &= -\sin x \cos y F(t) \\ p &= \frac{-\rho}{4}(\cos 2x + \cos 2y) F^2(t) \end{aligned} \tag{2}$$

where $F(t) = e^{-2\nu t}$ and $p$ is the pressure.

### 3.2 Results

Table 1 summarizes the performance of our model compared to the other PINN baselines as tested on the Taylor-Green vortex. RNN-SpDCT achieves the best performance for all values of vorticity used in the experiments. All RNN models achieve extremely accurate velocities compared to MLP-PINN. Figure 2 visualizes the predictions and compares with the analytical solution on the Taylor-Green vortex. The model produces much more accurate predictions for longer time steps (up to $2\pi$ seconds) compared to MLP-based PINNs.

## 4 Conclusion

We presented a novel extension to the PINN framework designed especially for time-dependent PDEs. Our model utilizes RNNs and DCTs to generate a multi-resolution latent context grid to condition the traditional MLP PINN architecture. We demonstrated that our model can accurately predict the solution functions in Taylor-Green vortex simulations (especially for pressures) and achieve state-of-the-art results. Future directions include experiments on more complex problems, higher dimensions, and longer time evolution.

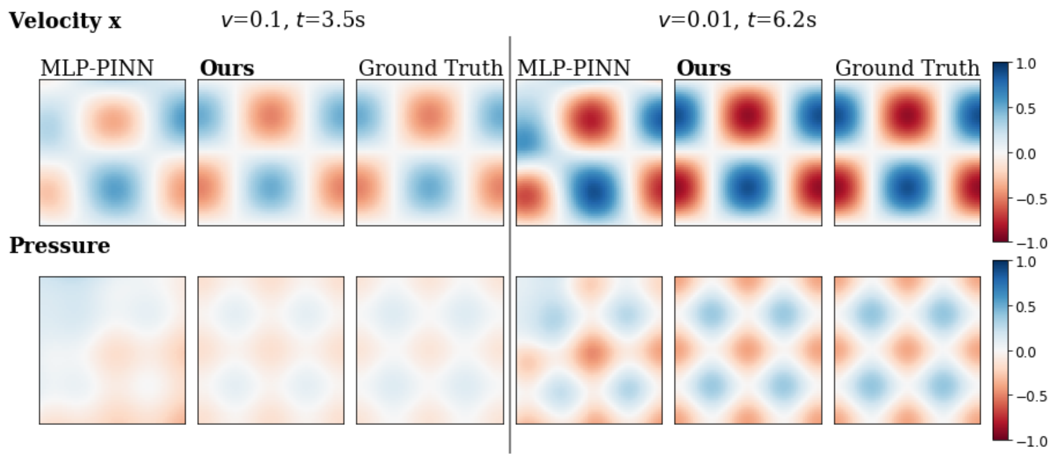

Figure 2: Visualization of the predictions on Taylor-Green vortex with the viscosity $\nu = 0.1$ at around 3.5 seconds (left) and $\nu = 0.01$ at around 6 seconds (right).

## Acknowledgments and Disclosure of Funding

This work was completed as part of the NVIDIA AI Research Residency Program. BW acknowledges support from a Grant-in-Aid for Scientific Research (KAKENHI Number 20K14706) of Japan Society for the Promotion of Science (JSPS).

## References

[1] Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[2] Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64(2):525–545, 2019.

[3] Oliver Hennigh. Lat-net: compressing lattice boltzmann flow simulations using deep neural networks. *arXiv preprint arXiv:1705.09036*, 2017.

[4] Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018.

[5] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2020.

[6] Bing Yu et al. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *arXiv preprint arXiv:1710.00211*, 2017.

[7] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL `https://www.sciencedirect.com/science/article/pii/S0021999118307125`.

[8] Leah Bar and Nir Sochen. Unsupervised deep learning algorithm for pde-based forward and inverse problems. *arXiv preprint arXiv:1904.05417*, 2019.

[9] Jonathan D Smith, Kamyar Azizzadenesheli, and Zachary E Ross. Eikonet: Solving the eikonal equation with deep neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, 2020.

[10] Kai Xu, Minghai Qin, Fei Sun, Yuhao Wang, Yen-Kuang Chen, and Fengbo Ren. Learning in the frequency domain. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1740–1749, 2020.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Soheil Esmaeilzadeh, Kamyar Azizzadenesheli, Karthik Kashinath, Mustafa Mustafa, Hamdi A Tchelepi, Philip Marcus, Mr Prabhat, Anima Anandkumar, et al. Meshfreeflownet: a physics-constrained deep continuous space-time super-resolution framework. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2020.

[13] Oliver Hennigh, Susheela Narasimhan, Mohammad Amin Nabian, Akshay Subramaniam, Kaustubh Tangsali, Zhiwei Fang, Max Rietmann, Wonmin Byeon, and Sanjay Choudhry. Nvidia simnet™: An ai-accelerated multi-physics simulation framework. In *International Conference on Computational Science*, pages 447–461. Springer, 2021.

[14] G. Taylor and A. Green. Mechanism of the production of small eddies from large ones. *Proceedings of The Royal Society A: Mathematical, Physical and Engineering Sciences*, 158:499–521, 1937.

## Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See section 3.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See section 3. We will add more detailed experimental setup in the final version.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See section 3
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See section 3
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]