REAL-TIME INFERENCE AND EXTRAPOLATION VIA A DIFFUSION-INSPIRED TEMPORAL TRANSFORMER OPERATOR (DITTO)

Oded Ovadia
Department of Applied Mathematics
Tel Aviv University
Tel Aviv, 69978, Israel
odedovadia@mail.tau.ac.il

Vivek Oommen
School of Engineering
Brown University
Providence, RI 02912, USA
vivek_oomen@brown.edu

Adar Kahana
Division of Applied Mathematics
Brown University
Providence, RI 02912, USA
adar_kahana@brown.edu

Ahmad Peyvan
Division of Applied Mathematics
Brown University
Providence, RI 02912, USA
ahmad_peyvan@brown.edu

Eli Turkel
Department of Applied Mathematics
Tel Aviv University
Tel Aviv 69978, Israel
eliturkel@gmail.com

George Em Karniadakis
Division of Applied Mathematics
Brown University
Providence, RI 02912, USA
george_karniadakis@brown.edu

ABSTRACT

Extrapolation remains a grand challenge in deep neural networks across all application domains. We propose an operator learning method to solve time-dependent partial differential equations (PDEs) continuously and with extrapolation in time without any temporal discretization. The proposed method, named Diffusion-inspired Temporal Transformer Operator (DiTTO), is inspired by latent diffusion models and their conditioning mechanism, which we use to incorporate the temporal evolution of the PDE, in combination with elements from the transformer architecture to improve its capabilities. Upon training, DiTTO can make inferences in real time. We demonstrate its extrapolation capability on a climate problem by estimating the temperature around the globe for several years, and also in modeling hypersonic flows around a double-cone. We propose different training strategies involving temporal-bundling and sub-sampling and demonstrate performance improvements for several benchmarks, performing extrapolation for long time intervals as well as zero-shot super-resolution in time.

Keywords Scientific machine learning · Diffusion models · Transformers · Partial differential equations.

1 Introduction

The field of scientific machine learning (SciML) has been growing rapidly in recent years, and many successful methods for modeling scientific problems using machine learning (ML) methods have been proposed [36, 28, 24, 27, 52]. Many tools designed for standard ML and data science problems can also perform well on SciML tasks. Recent innovations in the field of ML primarily originate from the domains of natural language processing [14] and computer vision [20]. Our work exploits and further develops the main idea of a recently proposed method called *diffusion models* [15] (used in generative AI) for solving forward partial differential equations (PDEs).

Solving time-dependent PDEs is an essential topic for the scientific community. If the underlying mathematical operators that govern the temporal evolution of the system are non-linear and/or there are observational data available, the task of assimilating and simulating such processes using discretization-based numerical methods can become increasingly challenging and computationally expensive. The burden associated with the traditional numerical solvers is further increased when separate simulation runs become mandatory for every new initial condition. Moreover, in certain application domains, such as autonomy and navigation or robotics, real-time inference is

required. SciML methods such as neural operators specifically address these issues by significantly reducing the associated computational costs [28, 24, 48, 3].

Several methods for solving PDE-related problems using ML methods, and specifically transformers, have been proposed [36, 28, 32]. We solve the forward problem of a time-dependent PDE by training the neural operator to accurately estimate the state field variable at a later time from a given initial condition. The main focus of this work is forecasting a continuous in-time solution in real-time for a plurality of initial conditions and extrapolating beyond the training domain. Solving PDE-related problems involves several challenges including generalizations for different problem conditions and dependence on the physical domain's discretization. To tackle the first, we utilize tools from the growing field of operator learning [24, 28], where we use learning techniques to map a function space to another one. Thus, we learn a family of solutions of PDEs corresponding to a family of initial conditions. For the second challenge, we propose a method that, while being dependent on the spatial discretization, is continuous in the temporal evolution of the solution, which is a prominent challenge in solving dynamical systems.

Recent works [35, 47] have demonstrated the efficiency of U-Net-based architectures for modeling time-dependent PDEs. However, the outputs of these U-Net-based architectures are discrete in time. Gupta *et al.* [7] performed a systematic ablation study to analyze the significance of Fourier layers, attention mechanism, and parameter conditioning in a U-Net-based neural operator. The DiTTO method proposed here is a diffusion-inspired model [45]. The common use of diffusion models involves a generative process used to create data samples. It incorporates a Markov chain of diffusion steps, where in each stage a different texture is added to the data sample. Usually, the texture is noise, so new noise distributions are incrementally added in each step. The models have also been used with other kinds of textures, for example creating cartoonish images from plain ones. Herein, we use a similar framework, but instead of conditioning on the noise distribution, we do so for the *temporal evolution.* We explore several implementations and training strategies, in addition to the diffusion models themselves. These enhanced methods form the class of explored DiTTO models. Importantly, we demonstrate how this framework can be used for extrapolation, i.e., it can make accurate predictions for samples outside the time interval it was trained to handle.

2 Methodology

We approximate the time evolution of a PDE solution (forward process). Instead of incrementally adding noise to the inputs, as done with diffusion models, we incrementally evolve the PDE solution over time. We replace the noise level parameter ε with the temporal variable t (see Appendix A.1, A.2 for more details). Then, we use the conditioning capabilities of diffusion models to learn the relations between the initial condition, the PDE solution,

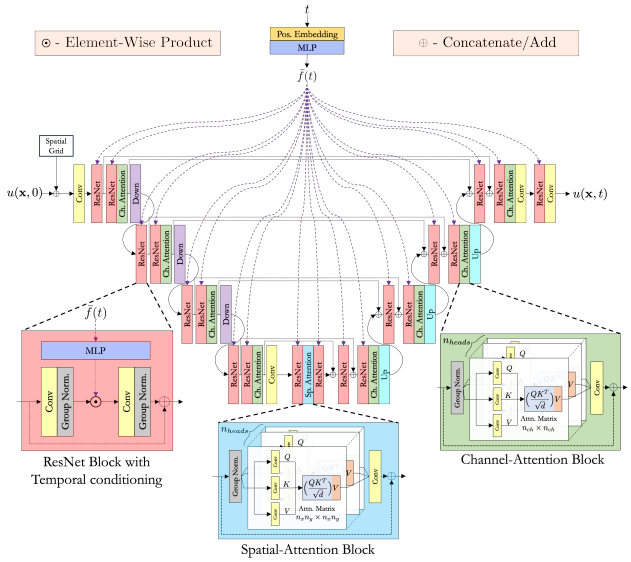


Figure 1: DiTTO architecture. The discretized initial condition $u(\mathbf{x}, 0)$ concatenated with the corresponding spatial grid, and the desired time $t \in \mathbb{R}^+$ are the respective inputs to the U-Net and the time-embedding network comprising DiTTO. The U-Net illustrated here consists of ResNet blocks with temporal conditioning, a Spatial-Attention block, and Channel-Attention blocks at 4 levels of coarseness and the corresponding residual connections across the same levels. The ResNet block conditions the non-linear representations of $u(\mathbf{x}, 0)$ with respect to the temporal embedding vector, $\vec{f}(t)$, by performing element-wise multiplication across the channels. Spatial-Attention and Channel-Attention blocks learn to extract correlations across space and channels, respectively.

and the time domain. After the training is complete, the model can interpolate between the initial and final time, creating a numerical solution that is continuous in time. We define its time evolution $\{x_t]\}$ as the following process:

$$\{x_t|\ t \in [0, t_{final}],\ x_t := u(\mathbf{x}, t)\}, \tag{1}$$

where u is the solution of the differential equation we attempt to solve. Using this notation, the operator learning problem becomes:

$$x_0 \longrightarrow x_t,\ \forall t \in [0, t_{final}], \tag{2}$$

$$x_t \approx \mathcal{G}(x_0)(t),\ \forall t \in [0, t_{final}], \tag{3}$$

where $\mathcal{G}$ represents the surrogate operator, DiTTO, where the operator learning technique we employ is the diffusion process. It is discrete, while (1) is continuous. We discretize $\{x_t\}$ by taking a partitioning $\{t_n\}_{n=0}^T$ of $[0, t_{final}]$, where $0 = t_0 < t_1 < \ldots < t_{T-1} < t_T = t_{final}$. The discrete process is then defined as $\{x_n\}_{n=0}^T$, where $x_n := u(\mathbf{x}, t_n)$. In PDE terms, given an initial condition x_0, we approximate the analytic solution at a set of specific future time steps $\{t_n\}_{n=1}^T$. In operator learning terms, we map a family of functions of the form $x_0 = u(\mathbf{x}, 0)$ to another family of functions of the form $u(\mathbf{x}, t)$.

The role of the neural network in diffusion models is to perform conditional denoising in each step. We repurpose this network structure to solve a PDE-related problem. Since $x_0, x_1, \ldots, x_T$ are taken from the analytical solution, there is no noise in this process. Therefore, there is no need for denoising. Thus, we replace the conditional denoising operation with a conditional temporal evolution $(x_0, t_n) \longrightarrow x_n$.

Next, we describe the DiTTO architecture. The network receives two main inputs: the initial condition $x_0 = u(\mathbf{x}, 0)$ and a time $t = t_n$. Recall that x_0 is a d-dimensional tensor, and t is a nonnegative scalar. For the temporal input t, we use an embedding mechanism based on the original Transformer positional encoding [49]. Each scalar t is mapped into a vector of size d_{emb}, and then passed through a simple multi-layer perceptron (MLP) with two linear layers and a GELU [13] activation function.

For the spatial input x_0, we concatenate it with a discrete spatial grid and provide it as an input to a U-Net [38]. We use a U-Net variant common in many diffusion models, such as DDPM [15]. It follows the backbone of PixelCNN++ [40], a U-Net based on a Wide ResNet [53, 12]. A sketch of the spatio-temporal architecture is given in Figure 1.

This architecture is not limited to a specific dimension. The same mechanism can be implemented for d-dimensional problems. The only major difference is the usage of d-dimensional convolutions for the relevant problem.

We extend DiTTO to develop three variants: DiTTO-s, DiTTO-point and DiTTO-gate. DiTTO-point is a memory-efficient version, and DiTTO-gate incorporates a gated sub-architecture motivated by Runge-Kutta methods. We also demonstrate the efficacy of two training strategies - 1) randomly sub-sampling the trajectory timesteps considered at each epoch (DiTTO-s), and 2) adopting the temporal-bundling [2] for enhancing the forecast capabilities. These extensions are further explained in the Appendix A.2.

3 Experiments and Results

We have tested the proposed DiTTO approach on various PDEs. Full details regarding these experiments are given in the appendix. Here, we present partial results for the hypersonics and climate cases in Figure 2. We compare DiTTO to another operator learning framework called Fourier neural operator (FNO) [24] when applicable.

3.1 Hypersonic Flow

We train DiTTO to learn the inviscid airflow around a double-cone object flying at a high Mach number. The flow physics at hypersonic Mach numbers around the double-cone geometry features complex transient events, stationary bow shock at the leading nose of the cone, and the interaction of the oblique shock wave originating from the leading edge with the bow shock formed around the upper part of the geometry of the cone. The shock layer is the narrow band between the wall of the double-cone geometry and the bow shock wave. In the shock layer, due to the interaction of shock waves, a triple point forms and generates vortical flow structures that move downstream. This interaction is challenging to capture using numerical solvers. The details regarding data creation and the governing equations are given in the Appendix B.

We now present two experiments conducted for this hypersonic problem. First, we learn the temporal evolution of the density field near the double cone. Specifically, we train the neural operator to learn the mapping from the incoming horizontal velocity field to the time-dependent density field around the double cone structure. The dataset comprises only 61 trajectories corresponding to Mach numbers $M \in [8, 10]$. It is split into training, validation, and testing datasets in the ratio 80:10:10. We use a cosine annealing type learning rate scheduler starting from 10^{-3} that decays to 0 during the training. We compare FNO, U-Net, DiTTO and DiTTO-s. Complete results are reported in the appendix. We observe in Figure 2A that DiTTO can accurately resolve the vortices close to the surface of the double cone.

Additionally, we perform an experiment where instead of conditioning on time as in most of the examples presented, we explore the ability to condition DiTTO on a different quantity: the Mach Number. We train the surrogate operator to learn the mapping from the density field at a specific time step and $M = 8$ to the density field at the same timestep but at different Mach numbers in the range $M \in [8, 10]$. We compare DiTTO with other neural operators. Our results suggest that DiTTO serves as an accurate surrogate conditioned on any scalar parameter and not necessarily time.

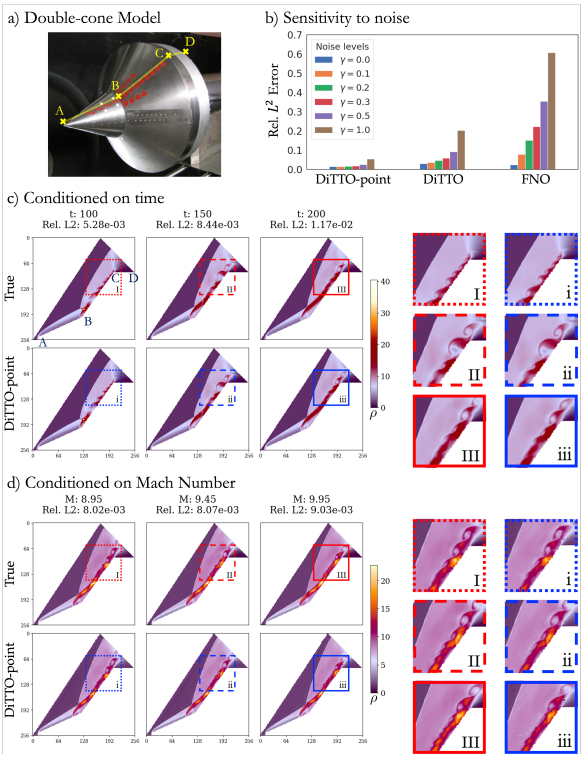
3.2 Climate Modeling

Climate models are complex, involving non-linear dynamics and multi-scale interactions between multiple variables. Describing climate behavior using PDEs is not a straightforward task. Consequently, there are many different numerical models and ML-based models [16, 1, 18, 33, 22, 30] that have been developed to model climate. Climate-related problems are challenging to approximate accurately at a low computational cost. Our goal is to efficiently learn the temporal evolution of the surface air temperature across the globe, and make accurate forecasts by extrapolating beyond the training period.

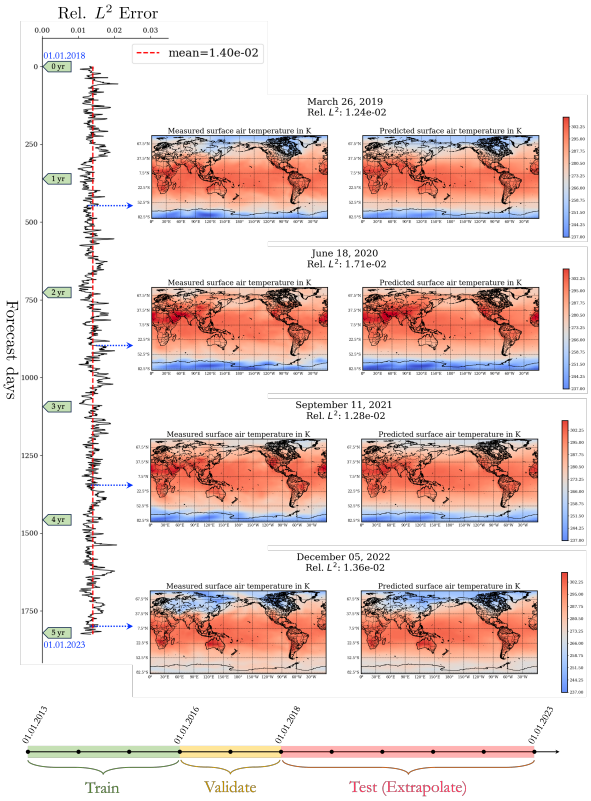
The results in Figure 2B demonstrate that DiTTO is able to extrapolate for long time intervals without any substantial accumulation of errors. The contour plots illustrate the measured (left) and predicted (right) global temperature profiles in Spring, Summer, Autumn, and Winter. DiTTO reduces the growth of the relative L^2 error across the 5 years of extrapolation with a mean of 0.014.

Acknowledgements

This work was supported by the Vannevar Bush Faculty Fellowship award (GEK) from ONR (N00014-22-1-2795). It is also supported by the U.S. Department of Energy, Advanced Scientific Computing Research program, under the Scalable, Efficient and Accelerated Causal Reasoning Operators, Graphs and Spikes for Earth and Embedded Systems (SEA-CROGS) project, DE-SC0023191.



(A) Modeling high-speed flow past a double-cone.



(B) Five-year climate forecast (extrapolation).

Figure 2: Two examples of DiTTO experiments.
Fig 2A: Modeling high-speed flow past a double-cone. a) The image of the double-cone model installed in the LENS XX tunnel [39]. b) The histogram illustrates the sensitivity of (i) DiTTO-point, (ii) DiTTO, and (iii) FNO to different noise levels γ. c) Temporal evolution of the density field (ρ) corresponding to test Mach number $(M) = 9.18$ around the double cone geometry predicted by DiTTO-point and its comparison with the solver simulation. d) Additional illustration of DiTTO-point conditioned on M and trained to predict the final state of ρ.
Fig 2B: Five-year climate forecast (extrapolation). DiTTO is trained on the global temperature data from years 2013 to 2015, validated from 2016 to 2017, and extrapolated from 2018 to 2022. The contour plots on the left and right columns correspond to snapshots from five years of measured and forecasted global temperature distribution, respectively. The four rows correspond to the global temperature profiles in Spring, Summer, Autumn, and Winter.

References

[1] Aniruddha Bora, Khemraj Shukla, Shixuan Zhang, Bryce Harrop, Ruby Leung, and George Em Karniadakis. Learning bias corrections for climate models using deep neural operators. *arXiv preprint arXiv:2302.03173*, 2023.

[2] Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers. *arXiv preprint arXiv:2202.03376*, 2022.

[3] Qianying Cao, Somdatta Goswami, and George Em Karniadakis. Lno: Laplace neural operator for solving differential equations. *arXiv preprint arXiv:2303.10528*, 2023.

[4] Shuhao Cao. Choose a transformer: Fourier or galerkin. *Advances in neural information processing systems*, 34:24924–24940, 2021.

[5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[6] Ruchi Guo, Shuhao Cao, and Long Chen. Transformer meets boundary value inverse problems. *arXiv preprint arXiv:2209.14977*, 2022.

[7] Jayesh K Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized pde modeling. *arXiv preprint arXiv:2209.15616*, 2022.

[8] Jacques Hadamard. Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton university bulletin*, pages 49–52, 1902.

[9] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. A survey on vision transformer. *IEEE transactions on pattern analysis and machine intelligence*, 45(1):87–110, 2022.

[10] Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. Gnot: A general neural operator transformer for operator learning. In *International Conference on Machine Learning*, pages 12556–12569. PMLR, 2023.

[11] Zhongkai Hao, Chengyang Ying, Zhengyi Wang, Hang Su, Yinpeng Dong, Songming Liu, Ze Cheng, Jun Zhu, and Jian Song. Gnot: A general neural operator transformer for operator learning. *arXiv preprint arXiv:2302.14376*, 2023.

[12] 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.

[13] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[14] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[15] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[16] James W Hurrell, Marika M Holland, Peter R Gent, Steven Ghan, Jennifer E Kay, Paul J Kushner, J-F Lamarque, William G Large, D Lawrence, Keith Lindsay, et al. The community earth system model: a framework for collaborative research. *Bulletin of the American Meteorological Society*, 94(9):1339–1360, 2013.

[17] Eugenia Kalnay, Masao Kanamitsu, Robert Kistler, William Collins, Dennis Deaven, Lev Gandin, Mark Iredell, Suranjana Saha, Glenn White, John Woollen, et al. The ncep/ncar 40-year reanalysis project. In *Renewable Energy*, pages Vol1_146–Vol1_194. Routledge, 2018.

[18] Georgios Kissas, Jacob H Seidman, Leonardo Ferreira Guilhoto, Victor M Preciado, George J Pappas, and Paris Perdikaris. Learning operators with coupled attention. *The Journal of Machine Learning Research*, 23(1):9636–9698, 2022.

[19] Nikola B. Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *CoRR*, abs/2108.08481, 2021.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 2012.

[21] Varun Kumar, Leonard Gleyzer, Adar Kahana, Khemraj Shukla, and George Em Karniadakis. Crunchgpt: A chatgpt assisted framework for scientific machine learning. *arXiv preprint arXiv:2306.15551*, 2023.

[22] Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Alexander Pritzel, Suman Ravuri, Timo Ewalds, Ferran Alet, Zach Eaton-Rosen, et al. Graphcast: Learning skillful medium-range global weather forecasting. *arXiv preprint arXiv:2212.12794*, 2022.

[23] Zijie Li, Kazem Meidani, and Amir Barati Farimani. Transformer for partial differential equations' operator learning. *arXiv preprint arXiv:2205.13671*, 2022.

[24] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[25] Xinliang Liu, Bo Xu, and Lei Zhang. Ht-net: Hierarchical transformer based operator learning model for multiscale pdes. *arXiv preprint arXiv:2210.10890*, 2022.

[26] Yue Liu, Zhengwei Yang, Zhenyao Yu, Zitu Liu, Dahui Liu, Hailong Lin, Mingqing Li, Shuchang Ma, Maxim Avdeev, and Siqi Shi. Generative artificial intelligence and its applications in materials science: Current situation and future perspectives. *Journal of Materiomics*, 2023.

[27] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning pdes from data. In *International conference on machine learning*, pages 3208–3216. PMLR, 2018.

[28] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021.

[29] Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022.

[30] Tung Nguyen, Johannes Brandstetter, Ashish Kapoor, Jayesh K Gupta, and Aditya Grover. Climax: A foundation model for weather and climate. *arXiv preprint arXiv:2301.10343*, 2023.

[31] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021.

[32] Oded Ovadia, Adar Kahana, Panos Stinis, Eli Turkel, and George Em Karniadakis. Vito: Vision transformer-operator. *arXiv preprint arXiv:2303.08891*, 2023.

[33] Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, et al. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.

[34] Ahmad Peyvan, Khemraj Shukla, Jesse Chan, and George Karniadakis. High-order methods for hypersonic flows with strong shocks and real chemistry. *Journal of Computational Physics*, 490:112310, 2023.

[35] Md Ashiqur Rahman, Zachary E Ross, and Kamyar Azizzadenesheli. U-no: U-shaped neural operators. *arXiv preprint arXiv:2204.11127*, 2022.

[36] Maziar Raissi, Paris Perdikaris, and George 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.

[37] Hendrik Ranocha, Michael Schlottke-Lakemper, Andrew Ross Winters, Erik Faulhaber, Jesse Chan, and Gregor Gassner. Adaptive numerical simulations with Trixi.jl: A case study of Julia for scientific computing. *Proceedings of the JuliaCon Conferences*, 1(1):77, 2022.

[38] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[39] Mehrnaz Rouhi Youssefi and Doyle Knight. Assessment of cfd capability for hypersonic shock wave laminar boundary layer interactions. *Aerospace*, 4(2):25, 2017.

[40] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

[41] Michael Schlottke-Lakemper, Gregor J Gassner, Hendrik Ranocha, Andrew R Winters, and Jesse Chan. Trixi.jl: Adaptive high-order numerical simulations of hyperbolic PDEs in Julia. `https://github.com/trixi-framework/Trixi.jl`, 09 2021.

[42] Michael Schlottke-Lakemper, Andrew R Winters, Hendrik Ranocha, and Gregor J Gassner. A purely hyperbolic discontinuous Galerkin approach for self-gravitating gas dynamics. *Journal of Computational Physics*, 442:110467, 06 2021.

[43] Dule Shu, Zijie Li, and Amir Barati Farimani. A physics-informed diffusion model for high-fidelity flow field reconstruction. *Journal of Computational Physics*, 478:111972, 2023.

[44] Chenyang Si, Ziqi Huang, Yuming Jiang, and Ziwei Liu. Freeu: Free lunch in diffusion u-net. *arXiv preprint arXiv:2309.11497*, 2023.

[45] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

[46] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[47] Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Daniel MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. Pdebench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems*, 35:1596–1611, 2022.

[48] Tapas Tripura and Souvik Chakraborty. Wavelet neural operator for solving parametric partial differential equations in computational mechanics problems. *Computer Methods in Applied Mechanics and Engineering*, 404:115783, 2023.

[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[50] Ting Wang, Petr Plechac, and Jaroslaw Knap. Generative diffusion learning for parametric partial differential equations. *arXiv preprint arXiv:2305.14703*, 2023.

[51] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: A survey. *arXiv preprint arXiv:2202.07125*, 2022.

[52] Hao Xu, Haibin Chang, and Dongxiao Zhang. Dl-pde: Deep-learning based data-driven discovery of partial differential equations from discrete and noisy data. *arXiv preprint arXiv:1908.04463*, 2019.

[53] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Supplementary Information

Appendix A: Detailed methods

A.1 Background

A.1.1 Operator learning

The standard use of ML models for scientific computations involves fitting a function to map numerical inputs to outputs. These inputs are ordinarily coordinates, materials, boundary conditions, etc., and the outputs are usually solutions of forward PDEs. An example is physics-informed neural networks (PINNs) [36], which use a deep neural network to solve PDEs by embedding elements from the PDE problem into the loss function. So, the network trains on the given data while using prior information about the problem it is solving. A major drawback is that for each problem, one needs to re-train the network, which is computationally expensive. This includes any changes to the parameters defining the problem.

Operator learning seeks to overcome this problem. Instead of fitting a function, one fits a mapping between two families of functions. Consider a generic family of d-dimensional time-dependent PDE problems of the form:

$$\begin{cases} \mathcal{L}u(\mathbf{x},t) = f(\mathbf{x},t), & \mathbf{x} \in D, t \in [0, t_{final}] \\ \mathcal{B}u(\mathbf{x},t) = g(\mathbf{x},t), & \mathbf{x} \in \partial D, t \in [0, t_{final}] \\ u(\mathbf{x},0) = I(\mathbf{x}), & \mathbf{x} \in D \end{cases}, \quad (4)$$

The differential operator $\mathcal{L}$ and forcing term f defines the PDE, the boundary operator $\mathcal{B}$ while g defines the solution on the boundary, t_{final} is the final physical time, I is the initial condition, and D is a Euclidean domain in $\mathbb{R}^d$ with boundary ∂D. We assume that the problem (4) is well-posed [8], so a unique solution exists.

Let $\mathcal{I}$ be a function space containing initial conditions of (4). Then there exists another space $\mathcal{U}$ that contains their respective solutions. We define an operator $\mathcal{G} : \mathcal{I} \longrightarrow \mathcal{U}$:

$$\mathcal{G}(I)(\mathbf{x},t) = u(\mathbf{x},t), \quad (5)$$

where $I \in \mathcal{I}, \mathbf{x} \in D$, and $t \in [0, t_{final}]$. Each initial condition $I \in \mathcal{I}$ is mapped into its corresponding solution $u \in \mathcal{U}$. The goal is to approximate the operator $\mathcal{G}$ using a neural network.

The first SciML operator learning method, called DeepONet, was proposed by Lu et al. [28]. The main components of a DeepONet are two neural networks: the branch and the trunk. Each network can be a fully connected neural network, convolutional, or any other architecture. Usually, the branch inputs are functions, and the trunk inputs are coordinates. DeepONets learn projections from the functions to a vector space, so they can map input functions to output functions at specific points.

Another operator learning approach is the Fourier neural operator (FNO) [24, 19]. FNOs, similarly to DeepONets, learn mappings between function spaces using projections. FNOs utilize the Fourier transform. They are effective and easy to implement,. FNOs are accurate, especially for smooth and periodic problems [29]. While the Fourier kernel is continuous it is necessary to use discrete versions for operator learning. Consequently, FNOs can be computationally costly when working with high-dimensional problems requiring many Fourier modes.

A.1.2 Transformers and attention

First presented by Vaswani et al. [49], transformers have been widely used in the ML community. Transformers introduce a new type of mechanism called the *scaled dot-product attention*. The attention module attempts to gather context from the given input. It operates on a discrete embedding of the data composed of discrete tokens.

The original architecture was proposed for natural language processing purposes, where one encodes sentences using their enumerated locations in the vocabulary. Since then, their usage has been extended to many other domains, and they outperform many different deep learning architectures in a wide variety of tasks. These domains include time series analysis [51] and computer vision [9]. For example, Vision Transformers (ViT) [5] split images into small patches, tokenize them, and apply the attention mechanism. In addition, they are computationally lighter than other alternatives and can be easily parallelized.

Transformers are becoming increasingly popular in the SciML community. They have been used for operator learning in many different ways [23, 25, 11, 10]. These methods show much promise by using the attention mechanism to find connections between points in the physical domain to function values. Some methods emphasize the attention mechanism itself [4, 6] and adapt it to PDE-related problems. Others utilize existing transformer models to solve PDE problems more easily [21]. In this work, we employ elements from the original Transformer architecture as part of the proposed neural network architecture.

A.1.3 Diffusion models

A diffusion model is a generative deep learning model that uses a Markov chain to produce samples that match a given dataset [45]. These models aim to learn the underlying distribution of a given dataset. After learning this distribution, they are used to generate new samples of similar properties to those found in the training set.

In [15], Ho et al. introduced a new type of diffusion model called denoising diffusion probabilistic models (DDPM). It consists of a forward diffusion process and an inverse one. In the forward case, Gaussian noise is incrementally added to the original sample for a given number of iterations. For a sufficiently large number of iterations, the noise completely destroys the original signal. Then, in the reverse diffusion process, the goal is to reconstruct the original signal by performing iterative denoising steps using a neural network. Diffusion models have been used

for SciML purposes, especially for generative artificial intelligence purposes [50, 43, 26]. While we are not using their generative capabilities in this work, we briefly explain their standard training procedure.

We present a mathematical formulation based on the works of Ho et al. [15] and Nichol et al. [31]. Given a data distribution $x_0 \sim q(x_0)$, we define a forward noising process q which produces steps $x_1, \dots, x_T$ by adding Gaussian noise at time t with variance $\beta_t \in (0, 1)$ as follows:

$$q(x_1, \dots, x_T | x_0) := \prod_{t=1}^{T} q(x_t | x_{t-1}), \tag{6}$$

$$q(x_t | x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t} x_{t-1}, \beta_t \mathbf{I}). \tag{7}$$

Given a sufficiently large T and a well-behaved schedule β_t, the latent x_T is nearly an isotropic Gaussian distribution. From (7), we see that x_t is drawn from a conditional Gaussian distribution with mean $\mu_t = \sqrt{1-\beta_t} x_{t-1}$ and variance $\sigma_t^2 = \beta_t$. In practice, this is done by randomly sampling a noise level parameter $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and setting:

$$x_t = \sqrt{1-\beta_t} x_{t-1} + \sqrt{\beta_t}\varepsilon. \tag{8}$$

Thus, if we know the exact reverse distribution $q(x_{t-1}|x_t)$, we can sample $x_T \sim \mathcal{N}(0, \mathbf{I})$ and run the process in reverse to get a sample from $q(x_0)$. However, since $q(x_{t-1}|x_t)$ depends on the entire data distribution, we approximate it using a neural network with hyperparameters θ as follows:

$$p_\theta(x_{t-1}|x_t) := \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)). \tag{9}$$

The neural network needs to learn the mean and variance to complete the backward diffusion process. Importantly, using the formulation in (8), in each step, it is sufficient to know β_t, x_t, and ε to approximate x_{t-1}. Then, the network is used autoregressively to reconstruct x_0. Assuming we know the schedules $\{\beta_t\}_{t=1}^T$, we can view the neural network as the following mapping:

$$(x_t, \varepsilon) \longrightarrow x_{t-1}. \tag{10}$$

In each step, the neural network performs a denoising operation, mapping x_t to a slightly less noisy signal x_{t-1}. Including the noise level parameter ε is essential for the denoising operation. During training, various noise levels are sampled. Knowing the specific noise level that distinguishes between consecutive states x_t and x_{t-1}, is crucial for effective denoising. Without this explicit knowledge of the noise level, the denoising process would become significantly more complicated, and the network may not converge. Thus we have a conditional denoising operation, conditioned on ε (or equivalently on the timestep with β_t).

A.2 DiTTO: Diffusion-inspired temporal transformer operator

As described in Methodology, we propose using temporal conditioning, instead of conditioning on noise as in Section A.1.3. We use the initial condition of a differential equation and infer the entire temporal process. It operates on various initial conditions, hence it is performing operator learning according to Section A.1.1. In addition, after training, the inference is possible on any real temporal value. Hence, the inference is continuous in time. As shown in the results, not only interpolation in time is possible, but also extrapolation.

To train the DiTTO network we gather data of multiple time-dependent procedures, varying in the initial condition. Then, we use the network architecture in Methodology, and select a training strategy (for example, temporal-bundling), to fit the data. We elaborate on these steps in the following subsections.

A.2.1 Training dataset

To train a neural network using the formulation presented in Methodology, we require a large set of initial conditions (inputs) and corresponding solutions (outputs). Let $\{I^m(\mathbf{x})\}_{m=1}^M$ be a set of initial conditions with corresponding analytic solutions $\{u^m(\mathbf{x}, t)\}_{m=1}^M$, where M is the desired number of training samples. Each sample consists of an initial condition and a PDE solution at the relevant timesteps. In practice, $\{u^m(\mathbf{x}, t)\}_{m=1}^M$ are numerical approximations of the analytic solutions and not analytic solutions which are often unavailable. Furthermore, the solutions are discretized in space using a grid that partitions the domain D. We emphasize that for all $m = 1, \dots, M$ and $t = 0, \dots, T$, $u^m(\mathbf{x}, t)$ is a matrix, and its dimensions depend on the spatial discretization parameters, i.e., the number of nodes along each axis.

We denote the forward process corresponding to the m-th initial condition and solution by $\{x_n^m\}_{n=0}^T$, where $x_n^m := u^m(\mathbf{x}, t_n)$. We define the following datasets:

$$\begin{aligned} \mathbf{X} &= \{(x_0^m, t_n) |\ n = 1, \dots, T, \quad m = 1, \dots, M\} \\ \mathbf{Y} &= \{x_n^m |\ n = 1, \dots, T, \quad m = 1, \dots, M\}. \end{aligned} \tag{11}$$

So, each solution of the PDE is transformed into T pairs of samples that correspond to the mapping described in Equation (3).

A.2.2 The DiTTO neural network architecture and parameters

We use the architecture described in Methodology. We mention that for the temporal input t, we use an embedding mechanism based on the original Transformer positional encoding [49]:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{emb}}}}\right),$$
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{(2i+1)/d_{\text{emb}}}}\right), \quad (12)$$

where d_{emb} is the desired embedding dimension. Each scalar t is mapped into a vector of size d_{emb}.

We now describe the loss function used as a target for training the network. Let $\mathcal{O}_\theta$ be the neural network described in Methodology with hyperparameters θ. The goal of $\mathcal{O}_\theta$ is to learn the mapping described in (3), using the dataset (11). We split this dataset into training, validation, and testing sets. We split them in a way that makes sure that no initial conditions from the validation and testing sets appear in the training set.

Diffusion models are often trained with a probabilistic loss function. However, since we learn a PDE operator, other loss functions commonly used for SciML applications are more fitting. Consequently, we train the network with a mean relative L^2 loss:

$$loss := \frac{1}{MT}\sum_{m=1}^{M}\sum_{n=1}^{T}\frac{||\mathcal{O}_\theta(x_0^m, t_n) - x_n^m||_2}{\varepsilon + ||x_n^m||_2}, \quad (13)$$

where ε is a small number used to prevent a zero denominator and stabilize the loss. The inputs and outputs of the model are d-dimensional, so they are converted into a one-dimensional array by column stacking (flattening) inside the loss function when needed. We describe the loss for the entire dataset for simplicity, but in practice, we divide it into batches.

Iterating over the entire dataset (11) can be time-consuming. For M_{train} initial conditions in the training set, we have $M_{train} \cdot T$ samples. So, the number of training steps scales linearly with T. So the number of training samples is very large for fine temporal discretizations.

A similar problem occurs in generative diffusion models. The original DDPM [15] requires hundreds of forward passes to produce good results. Later works suggested ways to improve the performance aspect of DDPMs. For example, Song et al. [46] suggest using non-Markovian processes to improve the computational cost. Nichol et al. [31] present a way to significantly reduce the number of necessary steps by sub-sampling the original diffusion process. Both methods focus primarily on the inference speed. However, in the case of DiTTO, inference is immediate. In Methodology, we explained that we do not view $x_0, x_1, \ldots, x_T$ as an iterative process. Instead, we treat each sample individually, significantly increasing the inference speed compared to generative models such as DDPM. Hence, we focus on speeding up the training process.

A.2.3 DiTTO-s

We propose DiTTO-s, a faster variant of DiTTO that relies on a sub-sampling mechanism similar to [31]. Instead of iterating over the entire process, we iterate over a random subsequence. Recall that for the m-th initial condition in the training set, the full process is $\{x_n^m\}_{n=1}^T$. Instead, we take a set of random subsequences $S_m \subset \{0, 1, \ldots, T\}$, such that $\sum_{m=1}^M |S_m| = \alpha MT$ for some $\alpha < 1$. Choosing $\alpha = 0.05$ means we only use 5% of the given samples in each epoch. The new DiTTO-s loss is given by:

$$loss_\alpha := \frac{1}{\alpha MT}\sum_{m=1}^{M}\sum_{n\in S_m}\frac{||\mathcal{O}_\theta(x_0^m, t_n) - x_n^m||_2}{\varepsilon + ||x_n^m||_2}, \quad (14)$$

After each epoch, we randomly sample S_m again using a uniform distribution. That way, given a sufficiently large number of epochs, we expect to cover a significant portion of samples in the dataset.

A.2.4 DiTTO-point

The architecture shown in Figure 1 can be used for problems in different dimensions. A way to accomplish that is to use d-dimensional convolutions in all the convolutional layers, where $d \in \{1, 2, 3\}$ is the physical dimension of the problem. Using high-dimensional convolutions requires a large amount of memory. Furthermore, many physical problems are not defined on structured grids. Thus, to use a neural operator framework, it is often necessary to project their solutions onto regular grids [29], which requires a change to the geometry of the problem.

To address these issues, we propose DiTTO-point, another variant of DiTTO that solves high-dimensional problems using exclusively 1-D convolutions. With DiTTO-point we treat the domain as a set of points in space instead of a structured d-dimensional grid. A domain with N points is defined as a $N \times d$ matrix, where each row represents a spatial coordinate. Similarly, we define the solution on this domain using a vector of size N, corresponding to the values of the solution at each point of the domain. Using this formulation, regardless of the original dimensionality of the problem, the input to DiTTO-point would always be of size $N \times d$. This enables the use of 1-D convolutions on this data, where d is the number of input channels.

However, directly using the architecture shown in Figure 1 with 1-D convolution does not work, without any modifications, on high-dimensional problems. Importantly, switching from a structured grid to a set of coordinates results in the loss of spatial information. This is especially true when the order in which the coordinates appear is not necessarily related to their physical distance. To solve this issue, we use another layer of positional encoding (see Equation (12)) based on the spatial coordinates of the grid. We apply this layer to the beginning of the overall architecture before we concatenate its output with the relevant initial condition in the latent space. This enables DiTTO-point to keep high-dimensional spatial information while using a one-dimensional architecture.

A.2.5 DiTTO-gate

We propose another variant of the architecture shown in Figure 1, called DiTTO-gate. Now, we modify the behavior of the skip connections of the U-Net decoder. Analysis of diffusion models shows that the U-Net skip connections introduce high-frequency features into the decoder [44]. For scenarios with fine details and sharp features, we put extra emphasis on the skip connections. So, we introduce a gate component, which operates directly on the skip connections. This component is composed of a standard convolutional block, as described in Methodology. So, in DiTTO-gate, we add such a component to each level of the decoder and use it on its incoming skip connection.

Appendix B: Hypersonic flow data generation and details

We first generate a training dataset by solving the 2D Euler equations on a fluid domain around a double-cone geometry. The governing equations are

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} = \mathbf{0}, \tag{15}$$

where the vector of conservative variables, x-direction and y-direction flux vectors, are described as

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix}, \mathbf{F} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(\rho E + p) \end{pmatrix}, \mathbf{G} = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ u(\rho E + p) \end{pmatrix}. \tag{16}$$

In Equation (15), t is time, x, y are the spatial coordinates, u and v are the x-direction and y-direction velocities, ρ is density, and p is the pressure. The total energy in Equation (16) is illustrated as

$$\rho E = \frac{p}{\gamma - 1} + \frac{1}{2}\rho(u^2 + v^2), \quad \gamma = 1,4 \tag{17}$$

We solve the system of equations using Trixi.jl numerical framework [41], which employs entropy stable discontinuous Galerkin spectral element (ES-DGSEM) approach [37, 42, 34] to solve hyperbolic and parabolic systems of equations. ES-DGSEM features high accuracy and stability and employs adaptive mesh refinement to automatically adapt the mesh resolution to high gradient regions of the flow field. The 2D physical domain is shown in Figure 3, where the boundary conditions are specified. Let M be the free-stream Mach number. The domain is initialized using constant uniform values for primitive variables as $u = M$, $v = 0$, $p = 1.0$, and $\rho = 1.4$. Each simulation is performed for a time $t \in [0, 0.04]$,. The solution values are shown at 201 snapshots corresponding to equidistance instances of time.

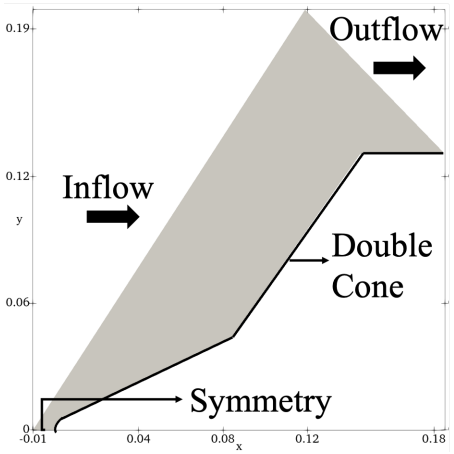


Figure 3: 2D domain of the double cone problem. The gray area is the domain and the highlighted lines show the surface of the double cone, which is a slip wall boundary condition. The small line at the bottom is a symmetry condition. The inflow and outflow boundaries are also shown.

Appendix C: Extrapolation in time

Extrapolation is a challenging problem due to the inherent nature of the data-driven surrogate networks to overfit the training distribution. We discuss a training strategy that partially alleviates the difficulties associated with extrapolation in time in using DiTTO.

During the training of DiTTO, the model is exposed only to the first 100 time steps. We train DiTTO with three types of time-series modeling strategies demonstrated in part a) of Figure 4 - i) autoregressive (look-forward window $lf = 1$), ii) temporal-bundling ($1 < lf < nt$) [2], and iii) mapping ($lf = nt$). We analyze their ability to extrapolate beyond the 100^{th} time step. Specifically, we train 6 different DiTTO models with lf=1, 5, 10, 20, 50, 100($= nt$). During the training, DiTTO learns a mapping $u(\bar{x}, t) \rightarrow u(\bar{x}, t + lf)$ such that $t + lf$ is less than the number of time steps, $nt = 100$. Hence, each trajectory in the training dataset is split into $nt - lf + 1$ sub-trajectories. During the inference stage, DiTTO leaps from $\hat{u}(\bar{x}, t)$ to $\hat{u}(\bar{x}, t + lf)$. Because the ground truth is not available, except at $t = 0$, we consider u as the ground truth and $\hat{u}$ as a prediction from DiTTO.

C.1 Navier-Stokes Equations

As an example, we train DiTTO on the two-dimensional incompressible Navier-Stokes equation for a viscous, incompressible fluid in vorticity form is given by:

$$\begin{cases} \partial_t \omega + u \cdot \nabla \omega = \nu \Delta \omega + f, & \\ \nabla \cdot u(x, y, t) = 0, & (x, y) \in (0, 1)^2, t \in (0, t_{final}] \\ \omega(x, y, 0) = \omega_0, & (x, y) \in (0, 1)^2 \end{cases} \tag{18}$$

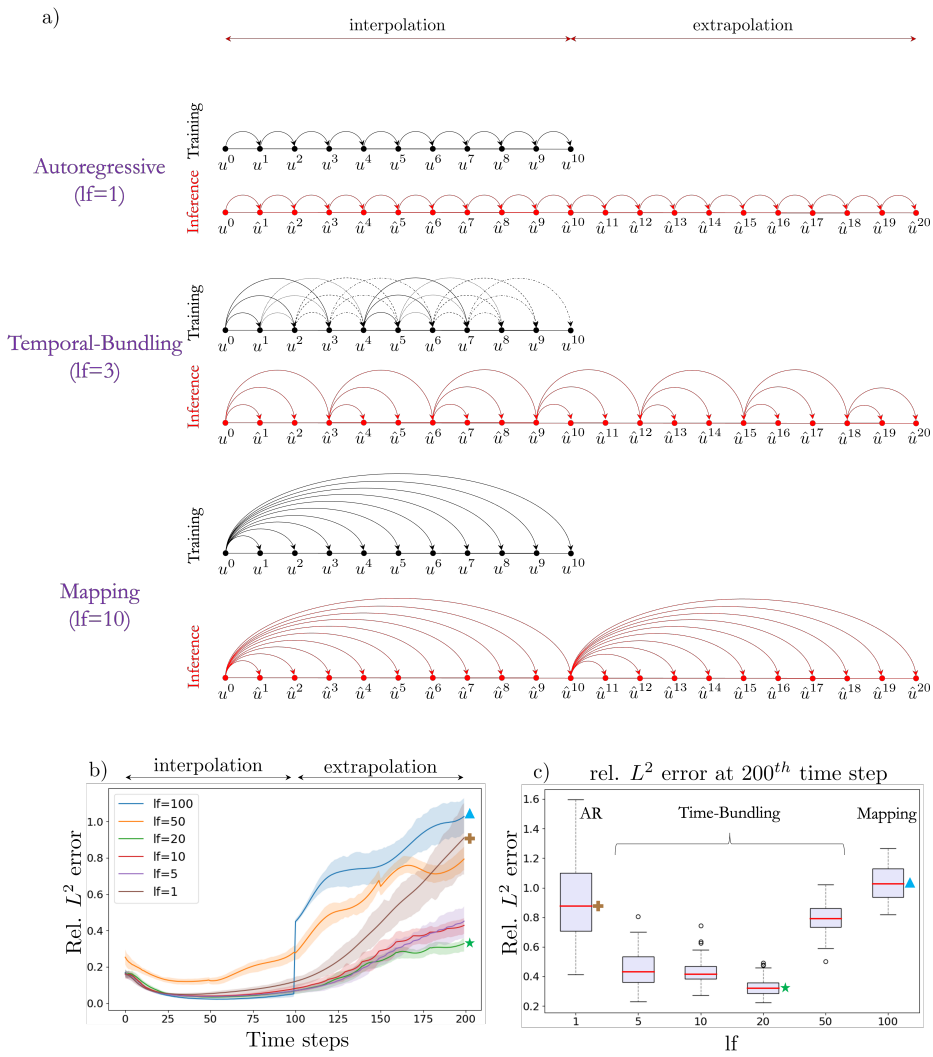


Figure 4: Temporal-bundling for efficient extrapolation. a) demonstrates 3 types of time-series modeling strategies for extrapolating beyond the training interval. We consider a time series with 20 time steps. b) visualizes the error accumulation for DiTTO models with different look-forward windows (lf) trained in the time interval 0-100 and extrapolated from 100-200. c) shows the relative percentage L^2 errors at the 200^{th} time step, corresponding to DiTTO models with different lf. The results in b) and c) come from the test dataset with previously unseen 100 initial conditions. The symbols in (b-c) denote the corresponding final time.

where ω is the vorticity, u is the velocity field, ν is the viscosity, and Δ is the two-dimensional Laplacian operator. We consider periodic boundary conditions. The source term f is given by $f(x,y) = 0.1(sin(2\pi(x+y)) + cos(2\pi(x+y)))$, and the initial condition $\omega_0(x)$ is sampled from a Gaussian random field according to the distribution $\mathcal{N}(0, 7^{3/2}(-\Delta + 49I)^{-5/2})$.

The dataset consists of 1000 trajectories with 200 time steps. It was randomly split into training, validation, and testing datasets in the ratio 80:10:10.

In Figure 4(b,c), we observe the minimum error with the lowest uncertainty occurs at $lf = 20$, suggesting that the temporal-bundling technique [2] offers a sweet spot between the autoregressive and the mapping strategies for extrapolating in time, with lower rates of error accumulation. Next, we investigate the extrapolation abilities of the temporal-bundling strategy for the climate problem.

C.2 Climate Dataset

We use a publicly available climate dataset provided by the Physical Sciences Laboratory meteorological data: `https://psl.noaa.gov/data/gridded/index.html` [17]. This data contains measurements of parameters over time, projected onto a spatial grid. We use the daily average surface temperature data (at 1000 millibar Pressure level) from January 1, 2013, to December 31, 2015, for training; January 1, 2016, to December 31, 2017, for validating; and January 1, 2018 to December 31, 2022

for testing (see Figure 2B). The data is projected to a spatial grid of dimensions 144×72, which corresponds to a resolution of 2.5° in both latitude and longitude. We split the data to training, validation and test sets in the ratio 30:20:50, for investigating the real-time long-term temporal extrapolation capability of DiTTO.

The experiments shown in Figure 4 motivated the adoption of the temporal-bundling technique with $lf = 365$ for training DiTTO. The training dataset can be interpreted as a single trajectory consisting of 1095 global temperature snapshots arising from the initial condition on Jan. 1, 2013. We utilize temporal-bundling with $lf = 365$ to split this trajectory into 730 sub-trajectories, each comprising 365 time steps. We further reduce the computational costs associated with the training and inference by performing a proper orthogonal decomposition and training DiTTO to learn the evolution of the eigen coefficients corresponding to the five dominant eigenvectors estimated from the global temperature profiles in the training dataset.