Physics-Informed Neural Operators with Exact Differentiation on Arbitrary Geometries

Colin White[*1], **Julius Berner**[*1], **Jean Kossaifi**[2], **Mogab Elleithy**[1], **David Pitt**[1]
Daniel Leibovici[1], **Zongyi Li**[1], **Kamyar Azizzadenesheli**[2], **Anima Anandkumar**[1,2]
[1] Caltech, [2] NVIDIA

Abstract

Neural Operators can learn operators from data, for example, to solve partial differential equations (PDEs). In some cases, this data-driven approach is not sufficient, e.g., if the data is limited, or only available at a resolution that does not permit resolving the underlying physics. The Physics-Informed Neural Operator (PINO) aims to solve this issue by adding the PDE residual as a loss to the Fourier Neural Operator (FNO). Several methods have been proposed to compute the derivatives appearing in the PDE, such as finite differences and Fourier differentiation. However, these methods are limited to regular grids and suffer from inaccuracies. In this work, we propose the first method capable of *exact* derivative computations for general functions on arbitrary geometries. We leverage the Geometry Informed Neural Operator (GINO), a recently proposed graph-based extension of FNO. While GINO can be queried at arbitrary points in the output domain, it is not differentiable with respect to those points due to a discrete neighbor search procedure. We introduce a fully differentiable extension of GINO that uses a differentiable weight function and neighbor caching in order to maintain the efficiency of GINO while allowing for exact derivatives. We empirically show that our method matches prior PINO methods while being the first to compute exact derivatives for arbitrary query points.

1 Introduction

Real-world science and engineering problems frequently involve partial differential equations (PDEs) [25], and existing numerical methods to solve them often require prohibitive computational resources [23]. Physics-Informed Neural Networks (PINNs) [8, 21] are machine learning methods to solve PDEs by using a neural network as the ansatz of the solution function. While PINNs have been successfully applied to several problems, their widespread application is limited by instabilities and optimization challenges [11]. The recently proposed Physics Informed Neural Operator (PINO) [17] could mitigate these issues and outperform PINNs. PINO is a *neural operator*, meaning that, unlike neural networks, it learns a map between function spaces. It is trained on multiple PDE instances, resulting in an easier optimization landscape compared to PINNs, which only solve a single instance.

Computing the PDE residual of a model output requires taking derivatives of that function. Several methods have been proposed to compute derivatives for PINO, such as finite differences, Fourier differentiation, and improvements of the latter using Fourier continuation [17, 19]. However, all of these methods suffer from discretization errors and cannot compute *exact* derivatives. A key challenge for PINO models based on the Fourier Neural

[*]Equal contribution. Correspondence to: {crwhite,jberner}@caltech.edu.

DLDE-III Wksp in the 37th Conference on Neural Information Processing Systems (NeurIPS 2023).

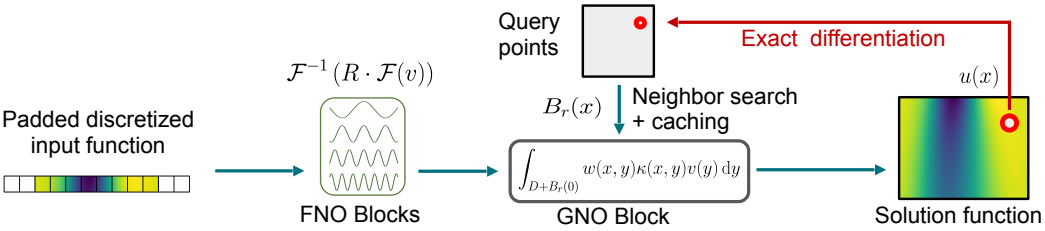

Figure 1: **Adding EDAM to PINO: Overview of our physics-informed GINO method using autograd**. A discretized and padded function is processed by a series of FNO blocks, which produces an output function on an extended regular grid. A GNO layer, modified to be differentiable, is used to query the output function at arbitrary points, and provide exact differentiation at that points. Note that irregular geometries can also be handled as inputs by first having a GNO layer before the FNO blocks.

Operator (FNO) [15] is that a trade-off must be made, either sacrificing exact derivatives, or the ability to query arbitrary output points.

In this work, we devise the first physics-informed neural operator capable of computing exact derivatives for arbitrary geometries. To accomplish this, we extend the Geometry-Informed Neural Operator (GINO) [12], a very recent extension of FNO that is able to handle arbitrary geometries. However, the output of GINO is not differentiable with respect to query coordinates due to a discrete neighbor search procedure. We introduce a fully differentiable extension of GINO that uses a differentiable weight function instead of the discrete neighbor search procedure, thus enabling the use of *automatic differentiation* (autograd). To preserve the efficiency of GINO, we use a neighbor caching algorithm to store the neighbors with nonzero weight. To also recover exact derivatives at the boundary, we extend the domain on which GINO operates using suitable domain padding. See Figure 1 for an overview. We empirically show that our method is on par with other PINO methods while being the first to allow querying arbitrary output points. We release our code at `https://anonymous.4open.science/r/exact-pino`.

2 Background and Related Work

PINN: Physics-informed neural nets (PINNs) [8, 21] are a family of deep learning methods whose loss function can include both agreement with training data points, as well as adherence to physics constraints such as from a PDE. PINNs have seen empirical success in learning a variety of PDEs [8, 21]. However, their training can be unstable and many follow-up works have attempted to improve the optimization behavior [7, 11, 27].

FNO: On the other hand, neural operators are a family of neural architectures that learn maps between function spaces rather than between finite-dimensional vector spaces [10], and they can be used to approximate the solution operator of a given PDE. In particular, the Fourier Neural Operator (FNO) has been successful in solving PDE-based problems with significant speedups [1, 5, 9, 15, 18]. A neural operator architecture can be defined as

$$\mathcal{G}_{\mathrm{FNO}} := \mathcal{Q} \circ (W_L + \mathcal{K}_L) \circ \cdots \circ \sigma\,(W_1 + \mathcal{K}_1) \circ \mathcal{P}, \tag{1}$$

where $\mathcal{P}$ and $\mathcal{Q}$ are pointwise lifting and projection operators, respectively. The intermediate layers consist of pointwise operators W_ℓ, integral kernel operators $\mathcal{K}_\ell$, and an activation function σ. In the case of the FNO, the integral kernel operators are linear transforms in Fourier space, $\mathcal{K}_\ell(v) = \mathcal{F}^{-1}(R_\ell \cdot \mathcal{F}(v))$, where R_ℓ are weight matrices, and $\mathcal{F}, \mathcal{F}^{-1}$ denote the Fourier transform and its inverse.

GINO: Very recently, the Geometry-Informed Neural Operator (GINO) [16] proposes to combine FNO with graph neural operators (GNO) [13, 14] in order to handle arbitrary geometries. Specifically, the output function v of $\mathcal{G}_{\mathrm{FNO}}$ in Equation (1) is processed with a graph operator,

$$\mathcal{G}_{\mathrm{GNO}}(v)(x) := \int_{B_r(x)\cap D} \kappa(x,y)v(y)\,\mathrm{d}y = \int_D \mathbb{1}_{B_r(x)}(y)\kappa(x,y)v(y)\,\mathrm{d}y, \tag{2}$$

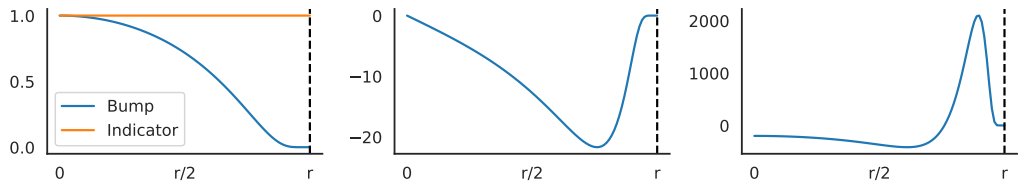

Figure 2: We show the weight function w in Equation (4) and its first and second derivatives (from left to right) as a function of $\|x - y\|$ for $r = 0.1$.

where $D \subset \mathbb{R}^d$ is the domain of v and $B_r(x)$ denotes a ball of radius $r \in (0, \infty)$ around $x \in D$. Note that the radius r is a hyperparameter and the integral is, for instance, approximated with a Riemann sum.

PINO: The physics-informed neural operator (PINO) [17] combines FNOs with the physics loss from PINNs. PINO has already been successfully applied to many PDEs [20, 22, 24]. Moreover, it was extended to solve coupled forward-backward PDEs [2] and coupled ODEs [3], and Fourier continuation has been proposed for better derivative computation [19]. Finally, we mention that DeepONets, another neural operator architecture, have also leveraged the PINN loss [6, 26, 28]. However, they can only be applied to a fixed input resolution.

3 Methodology

In this section, we present our new method, *Exact Derivatives on Arbitrary geoMetries* (EDAM): the first PINO capable of computing exact derivatives on arbitrary geometries. Recall from Equation (2) that the final layer in GINO computes an integral on a ball $B_r(x)$. This relies on an indicator function $\mathbb{1}_{B_r(x)}$, which is not differentiable with respect to x. Instead, we propose a fully-differentiable final layer, which replaces the indicator function $\mathbb{1}_{B_r(x)}$ by a differentiable weight function w as follows[2]:

$$\mathcal{G}_{\text{EDAM}}(v)(x) \coloneqq \int_{\widetilde{D}} w(x,y)\kappa(x,y)v(y)\,\mathrm{d}y, \quad x \in D. \tag{3}$$

Let us elaborate on the two new components, i.e., the weight function w and the extended domain $\widetilde{D} \coloneqq D + B_r(0) = \{x + y \colon x \in D,\ y \in B_r(0)\}$ in the next paragraphs.

Extended Domain: We pad the input function to the FNO, such that its output function v is supported on the extended domain $D + B_r(0)$. If we then query the GNO on the domain D, all kernel integrals are defined on the full ball $B_r(x)$; see Equation (2).

Weight function: We choose the radial weight function $w \colon \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$ given by

$$w(x,y) \coloneqq \mathbb{1}_{B_r(x)}(y) \exp\left(-\frac{r^2}{r^2 - \|x-y\|^2}\right), \tag{4}$$

see also Figure 2. Note that the definition of w is inspired by *mollifiers* in functional analysis (see, e.g., [4]), such that the indicator function is "smoothed out", i.e., w is infinitely often continuously differentiable. This allows us to compute the derivative

$$\partial_x \mathcal{G}_{\text{EDAM}}(v)(x) = \int_{\widetilde{D}} \partial_x \left[w(x,y)\kappa(x,y)\right] v(y)\,\mathrm{d}y = \int_{B_r(x)} \partial_x \left[w(x,y)\kappa(x,y)\right] v(y)\,\mathrm{d}y, \tag{5}$$

see Appendix A for the details. The expression in Equation (5) shows that we do not need to calculate the integral over the whole domain D, but can still rely on (a cached version of) neighbor search, which makes the method as efficient as the original GINO up to the negligible cost of evaluating the weight function w.

We emphasize that these definitions work for arbitrarily complex domains D. For regular grids, one could alternatively rely on, e.g., finite difference methods. However, it is well-known that such difference schemes suffer from the tradeoff between discretization error and catastrophic cancellation. We also find empirically that the derivatives using autograd are much smoother and more stable than the ones obtained with finite differences; see Figure 3.

[2]This can also be understood as considering the new kernel $\tilde{\kappa} \coloneqq \kappa w$.

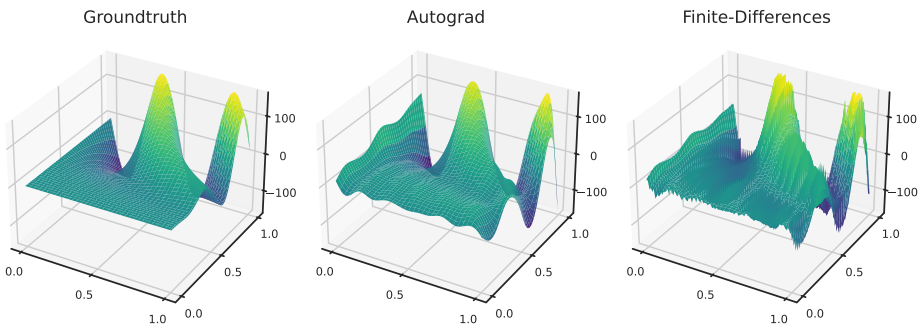

Figure 3: **Qualitative assessment of the smoothness of derivatives produced via PINO**. We fit GINO with PINO loss to the function $u\colon [0,1]^2 \to \mathbb{R}$, $(x, y) \to \sin(4\pi xy)$, and compare the derivatives $\partial_{xx} u$ of the **groundtruth** (*left*) with those obtained with **our approach** (*middle*) and **finite differences** (*right*).

Table 1: **Comparison of EDAM to other PINO methods on the Burgers' Equation.**

Method	**Full grid**		**Rand. subsampling**	
	Physics loss	L^2	**Physics loss**	L^2
FDM-PINO	5.19e-5	0.0358	N/A	N/A
FDM-Fourier-PINO	4.61e-5	0.0315	N/A	N/A
FC-PINO	9.14e-5	**0.0311**	N/A	N/A
EDAM-PINO [ours]	**8.67e-6**	0.0343	**1.03e-5**	**0.0361**

4 Experiments

We validate our method on the 1D time-dependent Burgers' equation with periodic boundary conditions, initial condition $u_0 \in L^2_{\text{per}}(D; \mathbb{R})$ with $D = (0, 1)$ and viscosity coefficient $\nu = 0.01$, see Appendix B. Our setting leads to the loss function

$$\mathcal{L}_{\text{PINO}}(v) = \left\| \partial_t v + \partial_x (v^2/2) - \nu \partial_{xx} v \right\|^2_{L^2(D\times(0,1))} + \alpha \left\| v(\cdot, 0) - u_0 \right\|^2_{L^2(D)}, \tag{6}$$

where $v = (\mathcal{G}_{\text{EDAM}} \circ \mathcal{G}_{\text{FNO}})(u_0)$. We found $\alpha = 10$ to work best for all considered methods.

We test EDAM as defined in the previous section, along with three different baselines for computing the PDE loss in PINO: the finite difference method [17]; a hybrid method that uses the finite difference method in the temporal dimension and Fourier differentiation in the spatial dimension [17]; and Fourier continuation for Fourier differentiation [19]. For all methods, we use the official implementation from prior work. To minimize confounding factors, we set the base architecture to GINO for all methods [16].

We run experiments with instance-wise fine-tuning as in prior work [19] (see Appendix C for full details). We train on a single initial condition for 1000 epochs, using only PDE loss, and we compare the final PDE loss and L^2 loss with the ground truth solution. For each experiment, we average the results over three initial conditions. First, we run all methods on a full, regular 128×26 grid, showing that EDAM-PINO outperforms all other method in terms of physics loss, and performs on par with all other methods in terms of L^2 loss. Next, we consider 3000 randomly sampled points from the domain $D \times (0, 1)$; EDAM-PINO is the only method that can handle such unstructured point clouds; see Table 1 for the results. While these points are sampled from a rectangular domain, EDAM-PINO can swiftly be applied to arbitrary geometries as well.

5 Conclusion and Future Work

We propose the first method for exact gradient computation in physics informed neural operators (PINOs) with arbitrary geometries. We empirically verify that our approach is on par with prior PINO methods while being the first to handle arbitrary output queries. Going forward, we will apply our method to irregular and large scale geometries.

References

[1] Boris Bonev, Thorsten Kurth, Christian Hundt, Jaideep Pathak, Maximilian Baust, Karthik Kashinath, and Anima Anandkumar. Spherical fourier neural operators: Learning stable dynamics on the sphere. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2023.

[2] Xu Chen, FU Yongjie, Shuo Liu, and Xuan Di. Physics-informed neural operator for coupled forward-backward partial differential equations. In *1st Workshop on the Synergy of Scientific and Machine Learning Modeling@ ICML2023*, 2023.

[3] Wenhao Ding, Qing He, Hanghang Tong, and Ping Wang. Pino-mbd: Physics-informed neural operator for solving coupled odes in multi-body dynamics. *arXiv preprint arXiv:2205.12262*, 2022.

[4] Lawrence C. Evans. *Partial differential equations*. American Mathematical Society, Providence, R.I., 2010.

[5] Vignesh Gopakumar, Stanislas Pamela, Lorenzo Zanisi, Zongyi Li, Anima Anandkumar, and MAST Team. Fourier neural operator for plasma modelling. *arXiv preprint arXiv:2302.06542*, 2023.

[6] Somdatta Goswami, Minglang Yin, Yue Yu, and George Em Karniadakis. A physics-informed variational deeponet for predicting crack path in quasi-brittle materials. *Computer Methods in Applied Mechanics and Engineering*, 391:114587, 2022.

[7] Zhongkai Hao, Jiachen Yao, Chang Su, Hang Su, Ziao Wang, Fanzhi Lu, Zeyu Xia, Yichi Zhang, Songming Liu, Lu Lu, et al. Pinnacle: A comprehensive benchmark of physics-informed neural networks for solving pdes. *arXiv preprint arXiv:2306.08827*, 2023.

[8] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[9] Jean Kossaifi, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, and Anima Anandkumar. Multi-grid tensorized fourier neural operator for high resolution PDEs, 2023.

[10] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

[11] Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34:26548–26560, 2021.

[12] Liam Li, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Geometry-aware gradient algorithms for neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[13] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.

[14] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 33:6755–6766, 2020.

[15] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, et al. Fourier neural operator for parametric partial differential equations. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[16] Zongyi Li, Nikola Borislavov Kovachki, Chris Choy, Boyi Li, Jean Kossaifi, Shourya Prakash Otta, Mohammad Amin Nabian, Maximilian Stadler, Christian Hundt, Kamyar Azizzadenesheli, and Anima Anandkumar. Geometry-informed neural operator for large-scale 3d pdes. *arXiv preprint arXiv:2309.00583*, 2023.

[17] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *arXiv preprint arXiv:2111.03794*, 2021.

[18] Burigede Liu, Nikola Kovachki, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, Andrew M Stuart, and Kaushik Bhattacharya. A learning-based multiscale method and its application to inelastic impact problems. *Journal of the Mechanics and Physics of Solids*, 158:104668, 2022.

[19] Haydn Maust, Zongyi Li, Yixuan Wang, Daniel Leibovici, Oscar Bruno, Thomas Hou, and Anima Anandkumar. Fourier continuation for exact derivative computation in physics-informed neural operators. *arXiv preprint arXiv:2211.15960*, 2022.

[20] Qinglu Meng, Yingguang Li, Xu Liu, Gengxiang Chen, and Xiaozhong Hao. A novel physics-informed neural operator for thermochemical curing analysis of carbon-fibre-reinforced thermosetting composites. *Composite Structures*, page 117197, 2023.

[21] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part I): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[22] Shawn G Rosofsky, Hani Al Majed, and EA Huerta. Applications of physics informed neural operators. *Machine Learning: Science and Technology*, 4(2):025022, 2023.

[23] Tapio Schneider, João Teixeira, Christopher S Bretherton, Florent Brient, Kyle G Pressel, Christoph Schär, and A Pier Siebesma. Climate goals and computing the future of clouds. *Nature Climate Change*, 7(1):3–5, 2017.

[24] Yuchen Song, Danshi Wang, Qirui Fan, Xiaotian Jiang, Xiao Luo, and Min Zhang. Physics-informed neural operator for fast and scalable optical fiber channel modelling in multi-span transmission. In *2022 European Conference on Optical Communication (ECOC)*, pages 1–4. IEEE, 2022.

[25] Gilbert Strang. Computational science and engineering. *Optimization*, 551(563):571–586, 2007.

[26] Sifan Wang and Paris Perdikaris. Long-time integration of parametric evolution equations with physics-informed deeponets. *Journal of Computational Physics*, 475:111855, 2023.

[27] Sifan Wang, Shyam Sankaran, Hanwen Wang, and Paris Perdikaris. An expert's guide to training physics-informed neural networks. *arXiv preprint arXiv:2308.08468*, 2023.

[28] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science advances*, 7(40):eabi8605, 2021.

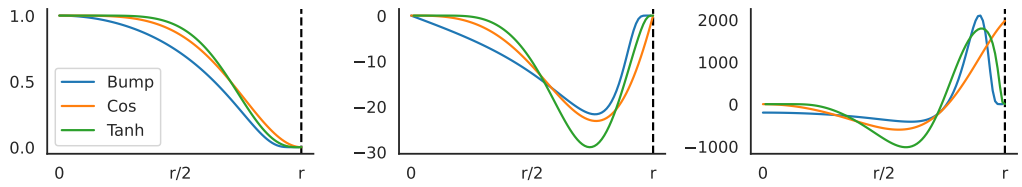

Figure 4: Comparison of the weight function w in Figure 2 with two other candidates based on rescaled and shifted cosine or hyperbolic tangent functions.

A Weight function

We first provide details on the derivation in Equation (5). In the first equality, we have used that the domain of integration does not depend on x anymore. Under suitable regularity assumptions, we can thus interchange integration and differentiation. The second equality follows by the definition of the weight function w in Equation (4), which implies that $\partial_x w = 0$ for $\|x - y\| > r$. Note that similar computations also hold for higher-order derivatives.

We have flexibility in choosing the weight function w and we also experimented with versions based on cosine and hyperbolic tangent functions; see Figure 4. However, we found that the function w in Equation (4) had the best numerical performance.

B Burgers' equation

The Burgers' equation models the propagation of shock waves and the effects of viscosity in fluid dynamics. We consider the 1D time-dependent Burgers' equation with periodic boundary conditions, initial condition $u_0 \in L^2_{\text{per}}(D; \mathbb{R})$ with $D = (0, 1)$ and viscosity coefficient $\nu = 0.01$. The goal is to learn the mapping from the initial condition to the solution, $\mathcal{G}^\dagger : u_0 \mapsto u\,|_{[0,1]}$, where

$$\begin{aligned} \partial_t u(x,t) + \partial_x(u^2(x,t)/2) &= \nu \partial_{xx} u(x,t), && x \in D,\ t \in (0,1] \\ u(x,0) &= u_0(x), && x \in D. \end{aligned}$$

We use the dataset from prior work, which consists of 800 instances with 128×100 resolution, where the initial condition is drawn from the Gaussian process $\mathcal{N}(0, 625(-\Delta + 25I)^{-2}$ [15, 17].

C Experimental Details

We test four PINO methods: EDAM; finite difference method [17]; a hybrid method that uses the finite difference method in the temporal dimension and Fourier differentiation in the spatial dimension [17]; and Fourier continuation for Fourier differentiation [19]. We use the official implementation of each baseline method. To minimize confounding factors, we set the base architecture to GINO for all methods [16].

The GINO consists of an FNO with four Fourier kernel layers, each with 16 frequency modes. Each intermediate weight matrix in the Fourier layers has a width of 26. We also learn the kernel coefficients in tensorized form as in prior work [9, 16], specifically with a Tucker factorization with rank 0.4. For GNO, we set the radius to 0.1, padding to 15, and the width of the weight matrix to 40.

We run instance-wise fine-tuning as in prior work [17, 19]. We train on a single initial condition for 1000 epochs, using only PDE loss. We use an initial learning rate of 0.0005, with a step function with $\gamma = 0.75$ every 100 epochs. We compare the final PDE loss and L^2 loss with the ground truth solution (averaged over three initial conditions).

For the complete experimental details, see our code at `https://anonymous.4open.science/r/exact-pino`.