Taylor Mode Neural Operators: Enhancing Computational Efficiency in Physics-Informed Neural Operators

Anas Jnini
University of Trento
anas.jnini@unitn.it

Flavio Vella
University of Trento
flavio.vella@unitn.it

Abstract

In this paper, we propose a novel application of Taylor-mode Automatic Differentiation (AD) to efficiently compute high-order derivatives in physics-informed neural operators (PINOs). Traditional approaches to automatic differentiation, particularly reverse-mode AD, suffer from high memory costs and computational inefficiencies, especially when dealing with high-order Partial Differential Equations (PDEs) and large-scale neural networks. Our method leverages Taylor-mode AD to forward-propagate Taylor series coefficients, enabling the efficient computation of high-order derivatives. We demonstrate our approach on two prominent neural operator architectures: DeepONets and Fourier Neural Operators (FNOs). Results indicate an order-of-magnitude speed-up over state-of-the-art methods for DeepONets and an eightfold acceleration for FNOs. Our code is publicly available at : https://github.com/HicrestLaboratory/Taylor-Mode-Neural-Operators.

1 Introduction

Partial Differential Equations (PDEs) are central to modeling physical systems in domains like fluid dynamics and material sciences. Traditional numerical solvers, such as finite element methods, often require fine discretization of the physical domain and a significant number of time-stepping iterations to achieve high accuracy, making them computationally expensive, especially for repeated solutions under varying conditions. Machine learning (ML) methods have recently emerged as promising alternatives, using neural networks (NNs) to approximate these complex mappings Kovachki et al. [2021], Li et al. [2020], Lu et al. [2019]. Neural operators, including DeepONets Lu et al. [2019] and Fourier Neural Operators (FNOs) Li et al. [2020], learn mappings between function spaces and offer advantages over traditional solvers in speed and scalability. However, they often require large datasets and may struggle with generalizing to unseen conditions Li et al. [2021a], Liu et al. [2022]. Physics-informed learning techniques, such as Physics-Informed Neural Networks (PINNs), embed physical laws directly into the learning process, reducing data requirements and improving generalization Raissi et al. [2019], Karniadakis et al. [2021], Jnini et al. [2024b]. The extension of this concept to neural operators—Physics-Informed Neural Operators (PINOs)—allows for more efficient and physically accurate solutions to PDEs Wang et al. [2021b].

Computing the loss functions for PDEs in both PINNs and PINOs, particularly for higher-order derivatives, typically relies on automatic differentiation (AD). However, AD, especially reverse-mode AD, can become computationally expensive due to the need to store intermediate variables (leaf nodes) and the exponential blowup in memory and computational cost when calculating high-order derivatives. This makes using standard AD methods prohibitive for large-scale networks or high-order PDEs. Taylor-mode Automatic Differentiation Bettencourt et al. [2019] has been proposed as an extension of forward-mode AD to efficiently compute high-dimensional derivatives, avoiding the exponential scaling of nested differentiation and significantly reducing computational costs. While it

Machine Learning and the Physical Sciences Workshop, NeurIPS 2024.

has been applied to train architactures like Multi-Layer Perceptrons (MLPs), its potential for use in neural operators architectures remains largely unexplored.

In this work, we leverage Taylor-mode Automatic Differentiation to reinterpret the computation graph of a neural operator's input derivatives as an expanded network with shared weights, enabling efficient computation of high-order derivatives at a cost proportional to a single forward pass. We derive specialized propagation rules for two well-known architectures—DeepONet and Fourier Neural Operators (FNOs)—demonstrating an order-of-magnitude speed-up for DeepONets and an eightfold speed-up for FNOs compared to state-of-the-art techniques, as shown in Section 5.

2 Related works

Efficient derivative computation is essential in scientific machine learning, especially when incorporating physical constraints into the loss function. Traditionally, reverse-mode AD is used in PINN training but incurs high memory costs due to storing all leaf nodes during the backward pass. To enhance AD efficiency, several methods have been proposed. For example, Separable PINNs Cho et al. [2023] reduce the number of leaf variables by separating variables for Multi-Layer Perceptrons (MLPs). Similarly, the Zero Coordinate Shift (ZCS) algorithm introduced by Leng et al. [2024] achieves state-of-the-art results for neural operators by simplifying derivative computation, reducing the number of leaf variables to one per spatial or temporal dimension. This method serves as a benchmark for our approach, as discussed in Section 5. Taylor-mode AD was initially introduced in PINNs by Wang et al. [2022] to handle high-order derivatives efficiently for MLPs and was used in Dangel et al. [2024] to derive an efficient KFAC approximation for PINNs. To our knowledge, this work is the first to apply Taylor-mode forward propagation to speed up the training of PINOs.

3 Background: Operator Learning

Operator learning focuses on approximating an operator $\mathcal{G} : X \to Y$ that maps between function spaces X and Y by employing a neural surrogate $\mathcal{G}_\theta$ such that $\mathcal{G}_\theta \approx \mathcal{G}$ Li et al. [2020]. A common application is the emulation of the solution map $\mathcal{G} : f \mapsto u$ for a given PDE, where $u = \mathcal{G}f$ represents the solution corresponding to the input data f.

Neural operators are generally trained using a regression-based approach, although it is possible to incorporate PDE information into the training process Li et al. [2021b]. In such cases, the loss function for the neural operator typically takes the form:

$$L(\theta) = \frac{1}{2} \sum_{i=1}^{N} \|\mathcal{G}_\theta(f_i) - u_i\|_Y^2 + \|\mathcal{L}(\mathcal{G}_\theta(f_i)) - f_i\|_Y^2,$$

where $(f_i, u_i)_{i=1,\dots,N}$ represents the training dataset, with the solutions $u_1, \dots, u_N$ typically generated by a classical numerical solver. The inclusion of the second term in the loss function can act as a regularization term in the absence of data, it leads to increased accuracy and generalization capability for the Neural Operators Wang et al. [2021b]. We will focus on two of the prominent neural operator architectures that are the deep operator network (DeepONet) Lu et al. [2019] and the Fourier neural operator (FNO) Li et al. [2020]. For an background overview of these architectures, we refer to Appendix 7.1.

4 Taylor-Mode Neural Operators(TMNO)

Taylor-mode automatic differentiation (AD) involves propagating Taylor series coefficients or directional derivatives through a neural network's computational graph. This technique effectively computes higher-order derivatives in Multi-Layer Perceptrons (MLPs) Wang et al. [2021a], Dangel et al. [2024]. An overview of Taylor-mode AD forward propagation rules for MLPs is provided in Appendix 7.2. A neural operator transforms an input function $v(x) \in V$, where V is a suitable function space, into an output function $u(x) = \mathcal{G}(v)(x) \in U$, where U is another function space, through a sequence of intermediate representations $\mathbf{z}^{(l)}(x) \in \mathbb{R}^{h^{(l)}}$ for each layer $l = 1, \dots, L$. Each intermediate representation $\mathbf{z}^{(l)}(x)$ can be viewed as a finite-dimensional projection or discretization of the underlying function defined on a domain $x \in D \subset \mathbb{R}^d$.

Taylor-Mode Propagation rule in DeepONet In DeepONet, the operator $\mathcal{G}(v)(\xi)$ is given by:

$$\mathcal{G}(v)(\xi) = \sum_{k=1}^{p} b_k(v) t_k(\xi) + b_0,$$

where $b_k(v)$ are outputs of the branch network and $t_k(\xi)$ are outputs of the trunk network. Using Taylor-mode AD, we propagate derivatives through the trunk network while treating the branch outputs as linear layers across function dimensions. For DeepONets, the input to the augmented trunk network consists of Taylor coefficients evaluated at collocation points of the output domain.First-order derivatives corresponding to the trunk inputs are set to standard basis vectors, while higher-order derivatives ($m > 1$) are initialized to zero.

Derivative Propagation rule through a DeepONet

The N-th order derivative with respect to ξ is computed as:

$$\frac{\partial^N \mathcal{G}(v)(\xi)}{\partial \xi_{i_1} \cdots \partial \xi_{i_N}} = \sum_{k=1}^{p} b_k(v) \sum_{P} \sigma^{(m)} \left(\mathbf{W}^{(L)} \mathbf{z}^{(L-1)} \right) \odot \prod_{p \in P} \left(\mathbf{W}^{(L)} \frac{\partial^{|p|} \mathbf{z}^{(L-1)}}{\prod_{j \in p} \partial \xi_j} \right).$$

Taylor-Mode Propagation rule in FNOs The Fourier Neural Operator (FNO) parameterizes the integral kernel in Fourier space, learning the Fourier coefficients of the output function for efficient computation. The input function $v(x)$ is lifted to a higher-dimensional representation $z_0(x) \in \mathbb{R}^{d_z}$ via a local transformation P. Iteratively, L Fourier layers are applied, each involving a Fast Fourier Transform (FFT), multiplication by a weight tensor R_l, and an inverse FFT:

$$\mathbf{z}_{l+1}(x) = \sigma \left(\mathcal{F}^{-1} \left(R_l \cdot \mathcal{F}(\mathbf{z}_l(x)) \right) + W_l \cdot \mathbf{z}_l(x) + \mathbf{b}_l \right),$$

where W_l and $\mathbf{b}_l$ are learnable parameters, and σ is a nonlinear activation function. The final output is projected back to the original space:

$$u(x) = Q(z_L(x)),$$

The FNO approximates solution operators for PDEs by combining FFT-based convolution with nonlinear transformations:

$$u(x) = Q \circ \mathcal{F}^{-1} \circ R_L \circ \mathcal{F} \circ \cdots \circ \mathcal{F}^{-1} \circ R_1 \circ \mathcal{F} \circ P(v(x)).$$

For an arbitrary dimensional FNO, we concatenate the computational grid, where the input functions are discretized, with the input functions themselves and feed them into the FNO. Due to the linearity of the Fast Fourier Transform (FFT), its inverse, and convolution, derivative propagation is handled similarly to linear layers. The augmented network receives Taylor coefficients of the input channels—either initialized to zero or computed numerically for PDE evaluation—concatenated with the grid coefficients. First-order derivatives corresponding to the grid input are set to standard basis functions, while higher-order derivatives ($m > 1$) are initialized to zero. Forward propagation of the k-th derivative is treated as an additional batch element, enabling parallel computation of higher-order derivatives. The main computational bottleneck arises in handling non-linearities, which depend on lower-order derivatives (see Appendix 7.3 for a visual overview).

Derivative Propagation through a Fourier Neural Operator (FNO) Layer

The N-th order derivative of the output $u(x)$ with respect to spatial coordinates is computed as:

$$\frac{\partial^N u(x)}{\partial x_{i_1} \cdots \partial x_{i_N}} = Q \circ \mathcal{F}^{-1} \circ R_L \circ \mathcal{F} \left(\frac{\partial^N \mathbf{z}_L(x)}{\partial x_{i_1} \cdots \partial x_{i_N}} \right),$$

where $\mathbf{z}_L(x)$ is the hidden state at the final Fourier layer L.

Derivative Propagation through Each FNO Fourier Layer

$$\frac{\partial^N \mathbf{z}_{l+1}(x)}{\partial x_{i_1} \cdots \partial x_{i_N}} = \sum_{P \in \text{Partitions}(N)} \sigma^{(|P|)}\left(\mathbf{z}_l(x)\right) \odot \prod_{p \in P} \left(\mathcal{F}^{-1} \left(R_l \circ \mathcal{F} \left(\frac{\partial^{|p|} \mathbf{z}_l(x)}{\prod_{j \in p} \partial x_j} \right) \right) + W_l \frac{\partial^{|p|} \mathbf{z}_l(x)}{\prod_{j \in p} \partial x_j} \right). \tag{1}$$

5 Experiments

In this section, we validate the effectiveness of our proposed TMNO approach on two challenging high-order PDEs: the 4th-Order Steady Streamline Navier-Stokes(**NSE**) equation for Kovasznay flow and the Kuramoto-Sivashinsky (**KS**) equation. Detailed hyperparameters are provided in Appendix 7.4. Our code is publicly available at : https://github.com/HicrestLaboratory/Taylor-Mode-Neural-Operators.

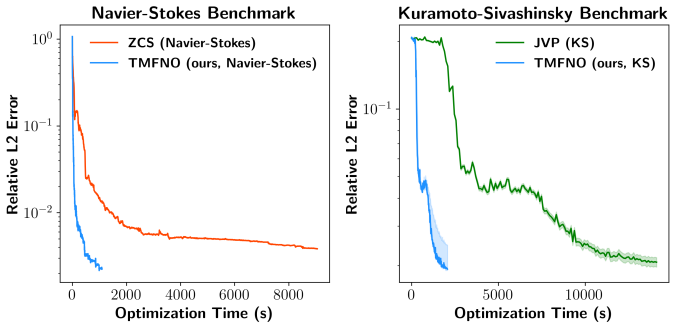

Figure 1: Evolution of relative L2 error over time for our method compared to benchmarked methods for both experiments accross five different seeds on validation data. The solid lines represent the median, and the shaded areas indicate the interquartile range.

4th-Order Steady Streamline NSE for Kovasznay Flow: The Kovasznay flow serves as a benchmark for the NS equations with analytical solutions. By employing a stream function $\psi(x, y)$ for a 2D incompressible fluid, the pressure term is eliminated, resulting in a 4th-order steady-state vorticity equation derived from the curl of the **NSE** equations. This formulation, given by

$$\frac{\partial \psi}{\partial y} \frac{\partial}{\partial x}(\nabla^2 \psi) - \frac{\partial \psi}{\partial x} \frac{\partial}{\partial y}(\nabla^2 \psi) = \nu \nabla^4 \psi,$$

where $\nu = \frac{\mu}{\rho}$ is the kinematic viscosity, ∇^2 is the Laplacian, and ∇^4 is the biharmonic operator, satisfies mass continuity and models the evolution of vorticity. We use a **DeepONet** trained with TMFNO and benchmark it against ZCS proposed in Leng et al. [2024]. The input to the branch net of the DeepONet is the Reynolds number at which the solution is being resolved, and the output is the streamline evaluated at collocation points. We train the model purely using the Physics-Informed Loss, without any trained labels. The input to the trunk net consists of the collocation points.

KS Equation: The KS equation, a fourth-order nonlinear PDE, models diffusive-thermal instabilities in a laminar flame front. It is expressed as

$$\frac{\partial u}{\partial t} + \frac{1}{2}\frac{\partial u^2}{\partial x} + \frac{\partial^2 u}{\partial x^2} + \frac{\partial^4 u}{\partial x^4} = 0,$$

with periodic boundary conditions $u(t,0) = u(t,L)$. Initial conditions are sampled as $u(x,0) = \sin\left(16\pi\frac{x}{L}\right) + 0.1\mathcal{N}(0,1)$, where $\mathcal{N}(0,1)$ denotes Gaussian noise. We employ a **FNO** trained with our Taylor-Mode Algorithm and benchmark it against a classical approach using nested Jacobian-vector products(JVP). In this case, the FNO maps initial conditions to spatio-temporal solutions and is trained with the Physics-informed Loss and data obtained from an Exponential Time Differencing solver. While this case can be handled with the analytical derivative proposed in Li et al. [2023], our example illustrates the speedup offered by our method for computing high-order derivatives, especially for non-periodic functions or those with shock waves.

Results and Discussion: Figure 1 shows that TMNO achieves a faster evaluation of the same derivative compared to benchmark methods. For the **NSE** , TMNO converges an order of magnitude faster than the ZCS method. For the **KS** equation, it similarly outperforms the JVP-based approach and achieves an eightfold speedup.

6 Conclusion

We introduced Taylor-Mode Neural Operators, an approach that treats the computation graph of neural operator input derivatives as a larger net with weight sharing, improving efficiency in derivative computations for physics-informed neural operators. Our method reduced the computational cost of training, achieving up to an order-of-magnitude speed-up for DeepONet architectures and an eightfold improvement for FNOs compared to current techniques. Future applications of interest will focus on optimizing parallelization of derivative computations and expanding the use of TMNO to more neural operator models and applications, notable applications in the context of Scientific Machine Learning could include the incorporation of Physical constraints into the architecture of the Neural Operators in line with the work proposed in Jnini et al. [2024a] and to speed-up the computation in second-order optimizers for Neural Operators Dangel et al. [2024], Jnini et al. [2024b], Jnini and Vella [2024].

References

Jesse Bettencourt, Matthew J. Johnson, and David Duvenaud. Taylor-mode automatic differentiation for higher-order derivatives in JAX. In *Program Transformations for ML Workshop at NeurIPS 2019*, 2019. URL https://openreview.net/forum?id=SkxEF3FNPH.

Junwoo Cho, Seungtae Nam, Hyunmo Yang, Seok-Bae Yun, Youngjoon Hong, and Eunbyung Park. Separable physics-informed neural networks, 2023. URL https://arxiv.org/abs/2306.15969.

Felix Dangel, Johannes Müller, and Marius Zeinhofer. Kronecker-factored approximate curvature for physics-informed neural networks, 2024. URL https://arxiv.org/abs/2405.15603.

Anas Jnini and Flavio Vella. Hessian-free natural gradient descent for physics informed machine learning, 2024. URL https://openreview.net/forum?id=Oqk1Ui6m0n.

Anas Jnini, Harshinee Goordoyal, Sujal Dave, Artem Korobenko, Flavio Vella, and Katharine Fraser. Physics-constrained deepONet for surrogate CFD models: a curved backward-facing step case. In *ICLR 2024 Workshop on AI4DifferentialEquations In Science*, 2024a. URL https://openreview.net/forum?id=zRef200Ucc.

Anas Jnini, Flavio Vella, and Marius Zeinhofer. Gauss-newton natural gradient descent for physics-informed computational fluid dynamics, 2024b. URL https://arxiv.org/abs/2402.10680.

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.

Nikola B Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

Kuangdai Leng, Mallikarjun Shankar, and Jeyan Thiyagalingam. Zero coordinate shift: Whetted automatic differentiation for physics-informed operator learning, 2024. URL `https://arxiv.org/abs/2311.00860`.

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.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021a. URL `https://openreview.net/forum?id=c8P9NQVtmnO`.

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*, 2021b.

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, 2023. URL `https://arxiv.org/abs/2111.03794`.

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.

Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

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.

Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021a.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets, 2021b. URL `https://arxiv.org/abs/2103.10974`.

Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality is all you need for training physics-informed neural networks. *arXiv preprint arXiv:2203.07404*, 2022.

7 Appendix

7.1 Background: Neural Operators

DeepONet DeepONet approximates operators by discretizing the input function v into a finite-dimensional space, sampling it at specific points $\{x_1, x_2, \ldots, x_m\}$. It consists of two sub-networks: the "trunk" network, which takes coordinates $\xi \in D'$ as input, and the "branch" network, which processes the sampled values of v. The operator is expressed as:

$$\mathcal{G}(v)(\xi) = \sum_{k=1}^{p} b_k(v) t_k(\xi) + b_0,$$

where b_0 is a bias term, $\{b_1(v), b_2(v), \ldots, b_p(v)\}$ are outputs of the branch network, and $\{t_1(\xi), t_2(\xi), \ldots, t_p(\xi)\}$ are outputs of the trunk network.

Fourier Neural Operator (FNO) The Fourier Neural Operator (FNO) parameterizes the integral kernel in Fourier space, learning the Fourier coefficients of the output function for efficient computation. The input function $v(x)$ is lifted to a higher-dimensional representation $z_0(x) \in \mathbb{R}^{d_z}$ via a local transformation P. Iteratively, L Fourier layers are applied, each involving a Fast Fourier Transform (FFT), multiplication by a weight tensor R_l, and an inverse FFT:

$$\mathbf{z}_{l+1}(x) = \sigma\left(\mathcal{F}^{-1}\left(R_l \cdot \mathcal{F}(\mathbf{z}_l(x))\right) + W_l \cdot \mathbf{z}_l(x) + \mathbf{b}_l\right),$$

where W_l and $\mathbf{b}_l$ are learnable parameters, and σ is a nonlinear activation function. The final output is projected back to the original space:

$$u(x) = Q(z_L(x)),$$

where Q is another local transformation. The FNO approximates solution operators for PDEs by combining FFT-based convolution with nonlinear transformations:

$$u(x) = Q \circ \mathcal{F}^{-1} \circ R_L \circ \mathcal{F} \circ \cdots \circ \mathcal{F}^{-1} \circ R_1 \circ \mathcal{F} \circ P(v(x)).$$

7.2 Taylor-Mode Automatic Differentiation for MLPs

Consider a multi-layer perceptron (MLP) $u_\theta = f_\theta^{(L)} \circ f_\theta^{(L-1)} \circ \cdots \circ f_\theta^{(1)}$ of depth $L \in \mathbb{N}$, where each layer $f_\theta^{(l)} \colon \mathbb{R}^{h^{(l-1)}} \to \mathbb{R}^{h^{(l)}}$ is parameterized by $\theta^{(l)} \in \mathbb{R}^{p^{(l)}}$. The network transforms an input $\mathbf{z}^{(0)} = \mathbf{x} \in \mathbb{R}^d$ into an output $u_\theta(\mathbf{x}) = \mathbf{z}^{(L)} \in \mathbb{R}^{h^{(L)}}$ through intermediate representations $\mathbf{z}^{(l)} \in \mathbb{R}^{h^{(l)}}$.

For Taylor-mode propagation, the initial conditions for derivatives of the input are:

$$\begin{aligned}
\frac{\partial \mathbf{z}^{(0)}}{\partial x_i} &= \mathbf{e}_i \in \mathbb{R}^d, \quad \text{(the } i\text{-th standard basis vector)}, \\
\frac{\partial^2 \mathbf{z}^{(0)}}{\partial x_i \partial x_j} &= \mathbf{0} \in \mathbb{R}^d.
\end{aligned}$$

For a linear layer $f_{\theta^{(l)}}(\mathbf{z}^{(l-1)}) = \mathbf{W}^{(l)}\mathbf{z}^{(l-1)}$, the derivatives propagate as:

$$\begin{aligned}
\mathbf{z}^{(l)} &= \mathbf{W}^{(l)}\mathbf{z}^{(l-1)}, \\
\frac{\partial \mathbf{z}^{(l)}}{\partial x_i} &= \mathbf{W}^{(l)} \frac{\partial \mathbf{z}^{(l-1)}}{\partial x_i}, \\
\frac{\partial^2 \mathbf{z}^{(l)}}{\partial x_i \partial x_j} &= \mathbf{W}^{(l)} \frac{\partial^2 \mathbf{z}^{(l-1)}}{\partial x_i \partial x_j}.
\end{aligned}$$

For a nonlinear activation function σ applied element-wise, the propagation rules are:

$$\begin{aligned}
\mathbf{z}^{(l)} &= \sigma(\mathbf{z}^{(l-1)}), \\
\frac{\partial \mathbf{z}^{(l)}}{\partial x_i} &= \sigma'\left(\mathbf{z}^{(l-1)}\right) \odot \frac{\partial \mathbf{z}^{(l-1)}}{\partial x_i}, \\
\frac{\partial^2 \mathbf{z}^{(l)}}{\partial x_i \partial x_j} &= \sigma''\left(\mathbf{z}^{(l-1)}\right) \odot \frac{\partial \mathbf{z}^{(l-1)}}{\partial x_i} \odot \frac{\partial \mathbf{z}^{(l-1)}}{\partial x_j} + \sigma'\left(\mathbf{z}^{(l-1)}\right) \odot \frac{\partial^2 \mathbf{z}^{(l-1)}}{\partial x_i \partial x_j}.
\end{aligned}$$

The general form for N-th order derivatives is given by Faà di Bruno's formula:

$$\frac{\partial^N \mathbf{z}^{(l)}}{\partial x_{i_1} \cdots \partial x_{i_N}} = \sum_{P \in \text{Partitions}(N)} \sigma^{(|P|)}\left(\mathbf{z}^{(l-1)}\right) \odot \prod_{p \in P} \frac{\partial^{|p|} \mathbf{z}^{(l-1)}}{\prod_{j \in p} \partial x_j},$$

7.3 Visual representation of Taylor-Mode Automatic Differentiation for FNOs

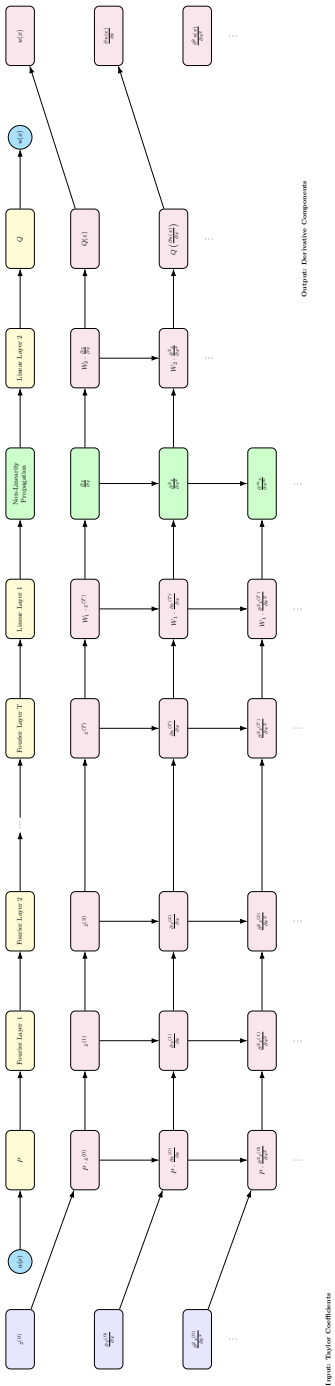

Figure 2: Diagram showing the forward propagation of derivatives through an FNO architecture using Taylor-mode automatic differentiation. The figure illustrates the input Taylor coefficients, their transformation through Fourier and linear layers, and the resulting derivative components. Each block in the figure represents a layer in the FNO architecture, with arrows indicating the propagation of derivatives of increasing order.

7.4 Hyperparameters for Kovasznay Flow with DeepONet

Table 1: Setup and Hyperparameters for Kovasznay Flow using DeepONet

Parameter	**Value**
Optimizers	L-BFGS
Architecture	DeepONet with 4-layer MLPs
Branch/Trunk Network Sizes	[100, 100, 100, 100]
Activation Function	Tanh
Random Seed	0,1,2,4,42
Domain	Kovasznay Flow in 2D incompressible fluid
Collocation Points (Interior)	2601
Number of Funcrions	200
Evaluation Points	9,000
Evaluation Metric	Relative L^2 error
Epochs	200,000 (L-BFGS)
Device	NVIDIA A100 80GB

7.5 Hyperparameters for Kuramoto-Sivashinsky Equation with FNO

Table 2: Setup and Hyperparameters for Kuramoto-Sivashinsky Equation using FNO

Parameter	**Value**
Optimizer	Adam
Architecture	FNO 2D
lAYERS	[64, 64, 64, 64]
Fourier Modes	[30, 30, 30, 30]
Final Layer Dimension	128
Activation Function	GELU
Random Seed	0,1,2,3,42
Spatial Points (nx)	512
Temporal Points (nt)	251
Time Horizon	1
Spatial domain	[0,1]
Samppes	100
Batch Size	1
Learning Rate	5×10^{-4}
Learning Rate Scheduler	StepLR (Step size: 20, Gamma: 0.5)
Epochs	1,150
Device	NVIDIA A100 80GB