# Time-Series Forecasting, Knowledge Distillation, and Refinement within a Multimodal PDE Foundation Model

Derek Jollie<sup>1,\*</sup>, Jingmin Sun<sup>2,\*</sup>, Zecheng Zhang<sup>3</sup>, and Hayden Schaeffer<sup>4</sup>

<sup>1</sup>Department of Mathematics, Montana State University, Bozeman, MT 59717.

<sup>2</sup>Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, PA 15213.

<sup>3</sup>Department of Mathematics, Florida State University, Tallahassee, FL 32304.

<sup>4</sup>Department of Mathematics, UCLA, Los Angeles, CA 90095.

<sup>\*</sup>The first two authors contributed equally to this work.

#### Abstract

Symbolic encoding has been used in multi-operator learning as a way to embed additional information for distinct time-series data. For spatiotemporal systems described by timedependent partial differential equations, the equation itself provides an additional modality to identify the system. The utilization of symbolic expressions along side time-series samples allows for the development of multimodal predictive neural networks. A key challenge with current approaches is that the symbolic information, i.e. the equations, must be manually preprocessed (simplified, rearranged, etc.) to match and relate to the existing token library, which increases costs and reduces flexibility, especially when dealing with new differential equations. We propose a new token library based on SymPy to encode differential equations as an additional modality for time-series models. The proposed approach incurs minimal cost, is automated, and maintains high prediction accuracy for forecasting tasks. Additionally, we include a Bayesian filtering module that connects the different modalities to refine the learned equation. This improves the accuracy of the learned symbolic representation and the predicted time-series.

# 1 Introduction

Operator learning, initially developed as an application of a universal approximation property in [2,3], aims to approximate maps between functions. Many mathematical and scientific problems can be formulated as the approximation of operators; for instance, forecasting time-series or solving time-dependent partial differential equations (PDEs). This has made operator learning a crucial tool in computational science and scientific machine learning (SciML) [11,13,14,20,23–28,37]. Many deep neural operators (DNOs) [12,13,15,20,21,32,39,40] have been developed and show effectiveness in solving different types of problems relating to time-dependent prediction. For example in [20], the authors introduced the Deep Operator Network (DeepONet) for approximating the solution map for ordinary differential equations (ODEs) and PDEs. In [8,16], the authors utilize DNOs to predict time-series recursively followed by numerical stabilization.

Though successful in many applications, a key challenge for DNOs is their limited ability to generalize, as they can only handle one operator at a time. To address generalization and extrapolation, i.e., the ability to predict new operators and time-series beyond the training

The code is available at: https://github.com/JingminSun/prose\_v1.



Figure 1: PROSE PDE Foundation Model with Particle Filtering.

interval, multi-operator learning (MOL) [18, 19, 29, 30, 33–35, 38] has been proposed. MOL uses a single network structure that is capable of processing data from multiple operators simultaneously. A crucial element of MOL is the way in which the identification of the system is encoded, since this guides the network toward understanding which operator is of interest for a given task, and provides a foundation for extrapolation to new operators. As a result, MOL networks trained with a diverse dataset and multiple modalities have become an approach for developing PDE foundation models.

Among the proposed MOL approaches, the first two notable contributions include PROSE [18, 19, 29, 30] and ICON [33, 35, 36]. PROSE is the first multimodal PDE foundation model that learns multiple operators and simultaneously predicts the equations that govern the physical system. It employs a symbolic encoding approach to provide additional information on the PDE of interest by embedding the equations into the feature space. Additionally, PROSE learns the governing physical system [24–26, 28, 31, 37] from the given data simultaneously as it constructs the evaluation operator used in forecasting. The learned systems are represented as time-dependent PDEs written by symbols and can be used to predict time-series beyond the training time interval.

Symbolic encoding has proven effective in various scenarios, including challenging extrapolation settings [29, 30]. Additionally, PROSE's symbolic encoding can be fine-tuned for downstream tasks and enables zero-shot prediction for new operators [30]. However, a challenge with symbolic encoding is that PDEs may be written in inconsistent orders and formats. For instance, the expressions x - 1 + 1 + y and y + x are mathematically equivalent but would be represented differently in token sequences. To address this issue, we propose a standardization approach that utilizes SymPy [22] for creating consistent token sequences for the symbolic modality. Our approach demonstrates effectiveness at automatically standardizing the token sequences and improving the encoding process.

To enhance the prediction accuracy of the physical system, we propose a sequential Monte Carlo (SMC) particle filter module [1, 5-7, 17] to refine the learned PDEs within the foundation model framework. Notably, only the coefficients of the PDEs require refinement, as PROSE has demonstrated reliability in correctly identifying the terms in the governing equation, i.e., terms such as  $u_{xx}$  and  $u_x$ . This also holds in the presence of noise or when terms are either missing or mistakenly included. The pipeline of our model is illustrated in Figure 1.

Our main contributions are as follows.

• We propose a new symbolic encoding method that can include a general equation modality. The new method allows the equations to be inputted without a specific format, thus leading to a more flexible model. Compared to the manual standardization methods, the proposed symbol encoding method significantly improves efficiency.

- We examine the ability of PROSE-PDE to generate consistent outputs when given incomplete symbolic inputs. In the experiments, we test the inclusion of placeholder coefficients on the equations and the addition of incorrect terms in the equations.
- A particle filter is introduced to the outputs of the decoders to further refine the learned coefficients, which leads to improved accuracy of the discovered equations. The refined model can be used for stable long-term predictions.

The code is available at https://github.com/JingminSun/prose\_v1.

#### 2 Methods

Suppose we are given data from  $N_{op}$  operators  $G_i: U_i \to V_i$ , where  $U_i$  and  $V_i$  are function spaces. MOL uses a single neural network  $\mathcal{G}_{\theta}$  to approximate  $G_i$ , i.e.,  $\mathcal{G}_{\theta}(G_i, u) \approx G_i(u)$ , where  $u \in U_i$  is a given input function for  $G_i$ ,  $i = 1, ..., N_{op}$ , and  $\theta$  is the network parameters. A key component of MOL is the encoding structure used to identify the system of interest as it informs the network of the particular PDE. Our focus is on PDE solution operators, which are crucial for many scientific computing problems. Therefore, we encode the governing equations directly to inform the network of  $G_i$ . PROSE introduces a symbolic encoding approach for this purpose. To encode the equations, PROSE represents each equation as a tree with nodes corresponding to operations and leaves to variables or coefficients. This tree is then converted into a sequence (in Polish notation), with each entry consisting of learnable tokens. For example,  $\cos(1.5x_1) + x_2^2 - 2.6$ is converted to sequence  $[+ \cos \times 1.5 x_1 - pow x_2 \ 2 \ 2.6]$ , where each entry is a trainable token. This is referred to as the PROSE tree.

PROSE's symbolic encoding proves effective even in challenging noisy extrapolation settings. However, the equations must be: (1) manually ordered into a particular format and (2) simplified to a standard expression. For example, the equation  $u_t - (u_x)_x = 0$  would need to be manually formatted as  $u_t - u_{xx} = 0$  to ensure that all tokens (operations and variables) fit within the existing library. This could lead to challenges in the testing phase when an equation is presented with a different order. It may become an issue for generalization, as determining the appropriate order when faced with new equations or terms can be difficult. Figure 2 is an example of the standard order used in PROSE and a possible alternative ordering. Notably, manually standardizing the tree for new equations with different orders can resolve the problem. However, the manual standardization process is time-consuming and costly.

To address these challenges, we first leverage SymPy to unify the expressions. This approach is



Figure 2: **PROSE Tree Examples:** The left tree is an example of a manually standardized PROSE tree for the viscous Burgers' equation  $u_t + kuu_x = \frac{\epsilon}{\pi}u_{xx}$ . In the experiments, to generate the randomized trees (or a tree encountered in testing), we randomly switch the order of any branch of the tree with probability 0.5, leading to different orders of the same symbolic expressions. The right tree is an example of an altered tree for the same equation.

both fast and cost-effective. Note that SymPy processes a mathematical equation using the same symbolic encoding procedure as PROSE but with an added tree-based transformation to simplify equations, i.e., the process is: equation-to-tree, simplified tree, sequence, and SymPy tokens. We refer to this as the *SymPy tree*. Notably, the family of SymPy tokens is larger than the useful tokens needed for encoding PDEs. To address this, we process the SymPy trees by simplifying unnecessary tokens, allowing the updated tokens to be used directly without manual adjustment. For instance, u(x,t) is tokenized as "u", "(", "x", ",", "t", and ")" in SymPy. We simplify this to "u(x,t)", significantly improving both efficiency and accuracy. Figure 3 is an illustration of the Korteweg–De Vries equation SymPy tree.

![](_page_3_Figure_1.jpeg)

Figure 3: SymPy Tree Example: KdV equation  $uu_x + u_t + \delta^2 u_{xxx} = 0$ . Here,  $\partial(u(x,t), (x,3))$  is used in the tree structure to embed the term  $u_{xxx}$  and similarly. Other derivatives are written using this notation.

**Bayesian Particle Filter**: We propose a refinement module that utilizes the SMC particle filter applied to the symbolic outputs to improve the accuracy of the learned equation. For a coefficient estimated by the symbolic decoder, we first set  $\alpha_0 := \alpha_{\text{PROSE}}$  and the initial distribution  $g_0(\alpha_0)$ for the particle filter to be a uniform distribution centered at  $\alpha_0$ , i.e.,  $g_0(\alpha_0) = \text{Unif.}(0.9\alpha_0, 1.1\alpha_0)$ . The parameter refinement update rule for  $\alpha_k$  is defined as:  $\alpha_k = \alpha_{k-1} + \nu$  where  $\nu \sim \mathcal{N}(0, \epsilon^2)$ is zero-mean Gaussian noise. Using the Chapman-Kolmogorov equation, we compute the prior belief distribution:

$$g_{k|k-1}(\alpha_k) = \int_{-\infty}^{\infty} f_{\nu}(\nu) g_{k-1}(\alpha_{k-1}) d\alpha_{k-1}$$
(1)

Bayes' theorem then gives the posterior belief with normal coefficient  $\eta$ :

$$g_k(\alpha_k) = p(\alpha_k | u(t_k, \cdot)) = \eta p(u(t_k, \cdot) | \alpha_k) g_{k|k-1}(\alpha_k).$$
(2)

This process is known as the Bayesian Filtering [4], and in practice, we implement it using a SMC particle filter simulation [1, 6, 7, 10, 17]. The details appear in Section 2.1 and Figure 4.

![](_page_3_Figure_8.jpeg)

Figure 4: Particle Filter Module: A discretized version of the Bayesian filter process.

#### 2.1 Particle Filter

In this section, we discuss a particle filter algorithm to approximate the distribution for  $\alpha$ . We can construct  $p(u(\cdot, t_k)|\alpha_k)$  from the evolution of u:

$$u(\cdot, t_k) = H(\alpha_k, u(\cdot, t_{k-1})) + \sigma$$
(3)

where H is a (deterministic) numerical scheme for solving the PDE and  $\sigma$  is the observation noise. In our case  $\sigma \sim f_{\sigma}(\sigma)$  is sampled from a zero-mean Gaussian distribution with variance  $\epsilon^2$ . Since  $\sigma$  is the source of randomness in the dynamics, inserting the distributions into Equation (2) yields:

$$g_k(\alpha_k) = \eta f_\sigma(\sigma) g_{k|k-1}(\alpha_k). \tag{4}$$

To compute this, we utilize a SMC particle method where we generate M particles and approximate (1) by:

$$g_{k|k-1}(\alpha_k) \approx \frac{1}{M} \sum_{i=1}^{M} \alpha_{k-1}^{(i)} + \nu = \frac{1}{M} \sum_{i=1}^{M} \alpha_{k|k-1}^{(i)}.$$
 (5)

Thus, we take the samples from the initial distribution of  $\alpha$  plus the process noise. Next, we calculate importance weights [9], which are given by:

$$w_i \propto \frac{g_k(\alpha^{(i)})}{g_{k|k-1}(\alpha_k)} = f_\sigma(\sigma), \qquad p_i = \frac{w_i}{w_0} \tag{6}$$

with normalization factor  $w_0 = \sum_{i=1}^{M} w_i$ . This allows for discrepancy between the distribution of interest and the distribution from which the samples are drawn. Finally, we compute the cumulative distribution function for  $\alpha$ :

$$G_{k}(\alpha) = \int_{-\infty}^{\alpha_{k}} g_{k}(\zeta) d\zeta$$
  
= 
$$\int_{-\infty}^{\alpha_{k}} \eta f_{\sigma}(\sigma) g_{k|k-1}(\zeta) d\zeta$$
  
$$\approx \frac{1}{\sum_{i=1}^{M} f_{\sigma}(\sigma)} \sum_{i=1}^{M} f_{\sigma}(\sigma) \mathbb{I}_{(-\infty,\alpha_{k})}(\alpha_{k|k-1}^{(i)})$$
(7)

$$=\sum_{i=1}^{M} p_{i} \mathbb{I}_{(-\infty,\alpha_{k})}(\alpha_{k|k-1}^{(i)})$$
(8)

where  $\mathbb{I}_A$  is the indicator function on the set A. Once this is constructed, we can resample  $\alpha_k$  from this new distribution  $G_k(\alpha)$ , then we repeat this process to construct the next step  $\alpha_{k+1}$ .

In this work, the particle-based refinement process uses M = 500 particles and uses 10 refinement steps. At the last step, we output the mean of  $G_{10}(\alpha)$  as the refined parameter, i.e.  $\alpha_{\text{refined}} = \frac{1}{M} \sum_{i=1}^{M} \alpha_{10}^{(i)}$ . The process noise is modeled as a normal distribution with variance equal to  $10^{-5}$ . Furthermore, the noise introduced by the numerical scheme is also modeled as Gaussian noise, with variance proportional to the initial  $L^2$ -norm of the state u, i.e.,  $\epsilon = 0.05 ||u(\cdot, t = 0)||_2$ .

#### 3 Numerical Experiments

We present numerical experiments to demonstrate that our proposed standardized symbol modality enhances prediction performance. We investigate five different symbolic encoding settings for PDEs and evaluate the trained model's performance on equations that are not preprocessed, i.e., not simplified or formatted in a specific order. The five settings used for testing (after pretraining) are: (1) PROSE Tree: defined in Section 2 and [29]; (2) Swapping PROSE tree: PROSE tree with randomized ordering for addition and subtraction (with -1 multiplied) with probability 0.5; (3) Noisy Swapping PROSE tree: PROSE tree with random erroneous terms added with probability 0.5, and the noisy trees are swapped with probability 0.5; (4) SymPy tree: defined in Section 2; and (5) Noisy SymPy tree: SymPy tree with random erroneous terms added with probability 0.5. We present an example of swapping terms and randomized ordering in Figure 2.

Table 1: **PROSE-PDE with Two Modalities.** Noisy: Erroneous terms in the input. Swapping: Rearranged order for terms.  $L^2$  and  $R^2$  errors are for the data predictions while Symbolic Error and Valid Fraction are metrics for the learned equations, see Appendix A.2 for details. PROSE Tree<sup>\*</sup> uses manual formatting and thus not a direct comparison.

Noise	Testing Tree Structure	Relative $L^2$ error	$R^2$ score	Symbolic Error	Valid Fraction
Noise-Free	PROSE Tree <sup>*</sup>	2.18%	0.995	1.24%	99.90%
	Swapping PROSE Tree	3.26%	0.983	1.43%	85.94%
	SymPy Tree	1.42%	0.996	1.40%	99.95%
Noisy Tree	Noisy Swapping PROSE Tree	4.53%	0.968	2.06%	76.01%
	Noisy SymPy Tree	3.81%	0.973	3.21%	83.23%

From Table 1, we observe that if the order of the terms in the testing equations does not match the training order, the errors increase to 3.26% and 1.43% respectively for the prediction and learned equations. In contrast, the SymPy tree achieves the best prediction errors at 1.42% and 1.40%, primarily due to the standardization of the format and the token library. This automated process is also faster compared to the manual standardization of the PROSE tree, which resulted in an error of 2.18%. Although the Symbolic Error increases between the Noisy PROSE tree and the Noisy SymPy tree, the valid fraction increases as well, showing more robust knowledge distillation. To further enhance the prediction accuracy of the physical system and utilize the learned equations to evaluate the time-series, we test the particle filter. Using the model obtained from the previous experiment with the SymPy tree, we randomly select 100 equations from each type for refinement. The results are presented in Table 2, with some corresponding predictions illustrated in Figure 5.

Table 2: Comparison of Symbolic Modality Errors with and without Particle Filter. We evaluate PDEs using the learned systems and calculate the Time-Series Errors (see Appendix A.2). (I)CL: (Inviscid) Conservation Law.

Type of equation	Expression	Symbolic Error		Time-Series Error	
Type of equation		Without Filtering	With filtering	Without Filtering	With filtering
Burgers'	$u_t + q_1(u^2)_x = q_2 u_{xx}$	1.02%	0.88%	1.50%	1.38%
Inviscid Burgers'	$u_t + q(u^2)_x = 0$	1.11%	0.65%	3.47%	2.12%
CL w. cubic flux	$u_t + q_1 (u^3)_x = q_2 u_{xx}$	3.07%	2.79%	3.94%	3.62%
ICL w. cubic flux	$u_t + q(u^3)_x = 0$	2.31%	1.73%	3.61%	2.97%
ICL w. sine flux	$u_t + q(\sin(u))_x = 0$	0.50%	0.29%	3.94%	2.22%

	PROSE Tree	Swapping PROSE Tree	Noisy PROSE Tree	Sympy Tree	Noisy Sympy Tree
Input Data					
Input Symbol	$[?]u_t + [?]\cos(u)u_x$	$[?]\cos(u)u_x + [?]u_t$	$[?]u_{xx} + [?]u_t + [?]\cos(u)u_x$	$[?]\cos(u)u_x+[?]u_t$	$[?]u + [?]\cos(u)u_x + [?]u_t$
Prediction					
Target Data					
Generated Symbol	$[1]u_t + [0.959]\cos(u)u_x$	$[1]u_t + [0.966]\cos(u)u_x$	$egin{aligned} [1]u_t + [0.931]\cos(u)u_x \ - [0.00318]u_{xx} \end{aligned}$	$[0.954]\cos(u)u_x + [1]u_t$	$[0.931]\cos(u)u_x + [1]u_t$
Refined Symbol	-	-	-	$[0.95451]\cos(u)u_x+[1]u_t$	$[0.95182]\cos(u)u_x+[1]u_t$

Figure 5: Various examples of the symbolic modality for inviscid conservation law with sine flux. Target equation:  $u_t + 0.955 \cos(u)u_x = 0$ . For PROSE tree, the model is trained for the order  $[?]u_t + [?]\cos(u)u_x$ , and for SymPy tree, the input expression is automatically uniformed into  $[?]\cos(u)u_x + [?]u_t$ . The generated symbols use 3 significant digits while the refinement is a standard float. Notably the SymPy tree removes the erroneous term in prediction. See Table 2 for error details.

# 4 Conclusion

In this work, we propose an automatic equation encoding modality for enhancing the time-series prediction of PDEs within the PROSE foundation model. This approach eliminates the need for costly manual ordering and simplification of PDEs, leading to significant improvements in prediction accuracy. To further refine the governing system learned by PROSE, we include a filter-based module that refines the learned expression. This refinement is possible due to the additional modality in the PDE foundation model. In future work, we will explore alternative refinement techniques to produce accurate and stable long-term predictions.

### Acknowledgments

This work was supported in part by NSF 2427558, NSF 2331033, and DE-SC0025440. The authors thank Yuxuan Liu from UCLA for his helpful comments and suggestions.

# References

- Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. Particle markov chain monte carlo methods. Journal of the Royal Statistical Society Series B: Statistical Methodology, 72(3):269–342, 2010.
- [2] Tianping Chen and Hong Chen. Approximations of continuous functionals by neural networks with application to dynamic systems. *IEEE Transactions on Neural networks*, 4(6):910–918, 1993.
- [3] Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.
- [4] Zhe Chen. Bayesian filtering: From kalman filters to particle filters, and beyond. Statistics, 182(1):1–69, 2003.
- [5] Randal Douc and Olivier Cappé. Comparison of resampling schemes for particle filtering. In ISPA 2005. Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, 2005., pages 64–69. Ieee, 2005.
- [6] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and computing*, 10:197–208, 2000.
- [7] Arnaud Doucet, Adam M Johansen, et al. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of nonlinear filtering*, 12(656-704):3, 2009.
- [8] Yalchin Efendiev, Wing Tat Leung, Guang Lin, and Zecheng Zhang. Efficient hybrid explicitimplicit learning for multiscale problems. *Journal of Computational Physics*, page 111326, 2022.
- [9] John Geweke. Bayesian inference in econometric models using monte carlo integration. Econometrica: Journal of the Econometric Society, pages 1317–1339, 1989.
- [10] Fredrik Gustafsson. Particle filter theory and practice with positioning applications. *IEEE Aerospace and Electronic Systems Magazine*, 25(7):53–82, 2010.
- [11] 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.
- [12] Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries. arXiv preprint arXiv:2207.05209, 2022.
- [13] 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.
- [14] 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.
- [15] Guang Lin, Christian Moya, and Zecheng Zhang. B-deeponet: An enhanced bayesian deeponet for solving noisy parametric pdes using accelerated replica exchange sgld. *Journal* of Computational Physics, 473:111713, 2023.

- [16] Guang Lin, Christian Moya, and Zecheng Zhang. Learning the dynamical response of nonlinear non-autonomous dynamical systems with deep operator neural networks. *Engineering Applications of Artificial Intelligence*, 125:106689, 2023.
- [17] Guang Lin, Zecheng Zhang, and Zhidong Zhang. Theoretical and numerical studies of inverse source problem for the linear parabolic equation with sparse boundary measurements. *Inverse Problems*, 38(12):125007, 2022.
- [18] Yuxuan Liu, Jingmin Sun, Xinjie He, Griffin Pinney, Zecheng Zhang, and Hayden Schaeffer. PROSE-FD: A multimodal pde foundation model for learning multiple operators for forecasting fluid dynamics. arXiv preprint arXiv:2409.09811, 2024.
- [19] Yuxuan Liu, Zecheng Zhang, and Hayden Schaeffer. Prose: Predicting operators and symbolic expressions using multimodal transformers. arXiv preprint arXiv:2309.16816, 2023.
- [20] 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.
- [21] 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.
- [22] Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, 2017.
- [23] 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.
- [24] Hayden Schaeffer. Learning partial differential equations via data discovery and sparse optimization. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 473(2197):20160446, 2017.
- [25] Hayden Schaeffer, Russel Caflisch, Cory D Hauck, and Stanley Osher. Sparse dynamics for partial differential equations. *Proceedings of the National Academy of Sciences*, 110(17):6634– 6639, 2013.
- [26] Hayden Schaeffer and Scott G McCalla. Sparse model selection via integral terms. Physical Review E, 96(2):023302, 2017.
- [27] Hayden Schaeffer and Stanley Osher. A low patch-rank interpretation of texture. SIAM Journal on Imaging Sciences, 6(1):226–262, 2013.
- [28] Hayden Schaeffer, Giang Tran, and Rachel Ward. Extracting sparse high-dimensional dynamics from limited data. SIAM Journal on Applied Mathematics, 78(6):3279–3295, 2018.
- [29] Jingmin Sun, Yuxuan Liu, Zecheng Zhang, and Hayden Schaeffer. Towards a foundation model for partial differential equations: Multi-operator learning and extrapolation. arXiv preprint arXiv:2404.12355, 2024.
- [30] Jingmin Sun, Zecheng Zhang, and Hayden Schaeffer. Lemon: Learning to learn multi-operator networks. arXiv preprint arXiv:2408.16168, 2024.

- [31] Yifan Sun, Linan Zhang, and Hayden Schaeffer. Neupde: Neural network based ordinary and partial differential equations for modeling time-dependent data. In *Mathematical and Scientific Machine Learning*, pages 352–372. PMLR, 2020.
- [32] Gege Wen, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, and Sally M Benson. U-fno—an enhanced fourier neural operator-based deep-learning model for multiphase flow. Advances in Water Resources, 163:104180, 2022.
- [33] Liu Yang, Siting Liu, Tingwei Meng, and Stanley J Osher. In-context operator learning with data prompts for differential equation problems. *Proceedings of the National Academy of Sciences*, 120(39):e2310142120, 2023.
- [34] Liu Yang, Tingwei Meng, Siting Liu, and Stanley J Osher. Prompting in-context operator learning with sensor data, equations, and natural language. arXiv preprint arXiv:2308.05061, 2023.
- [35] Liu Yang and Stanley J Osher. Pde generalization of in-context operator networks: A study on 1d scalar nonlinear conservation laws. arXiv preprint arXiv:2401.07364, 2024.
- [36] Yibo Yang, Georgios Kissas, and Paris Perdikaris. Scalable uncertainty quantification for deep operator networks using randomized priors. Computer Methods in Applied Mechanics and Engineering, 399:115399, 2022.
- [37] Linan Zhang and Hayden Schaeffer. On the convergence of the sindy algorithm. Multiscale Modeling & Simulation, 17(3):948–972, 2019.
- [38] Zecheng Zhang. Modno: Multi-operator learning with distributed neural operators. Computer Methods in Applied Mechanics and Engineering, 431:117229, 2024.
- [39] Zecheng Zhang, Christian Moya, Lu Lu, Guang Lin, and Hayden Schaeffer. D2no: Efficient handling of heterogeneous input function spaces with distributed deep neural operators. *Computer Methods in Applied Mechanics and Engineering*, 428:117084, 2024.
- [40] Zecheng Zhang, Leung Wing Tat, and Hayden Schaeffer. Belnet: basis enhanced learning, a mesh-free neural operator. Proceedings of the Royal Society A, 479(2276):20230043, 2023.

# A Experiment Setup

#### A.1 Dataset

The dataset utilizes the conservation laws from [29]. To summarize, it consists of 6 families of conservation laws: Inviscid/ viscous Burgers', inviscid/ viscous conservation law with cubic flux, and inviscid/ viscous conservation law with sine flux. The parameters are randomly sampled from  $\pm 10\%$  of the original value and 50 initial conditions leading to 153.6K separate equations used in training. Then 30.72K equations with different parameters are used for testing.

The initial data sequence is obtained from the PDE dataset using 16 timestamps from  $[0, t_f/2]$ ( $t_f$  specified per equation) with 128 points for the spatial grid on  $[0, x_f]$  for a fixed  $x_f$ . Note that a change of variables is used to re-scale and normalize the PDEs so that their solutions reside on a specified interval. We perform data normalization during the training process. Given the data input sequence  $\{u(t_i, \cdot)\}_{0 \le i < T_0}$ , we compute the mean and standard deviation, which are used to normalized both the input and ground truth label. The loss function is the standard mean squared error in this normalized space.

#### A.2 Evaluation Metrics

Since we use two modalities, we utilize four evaluation metrics from [29]. For metrics on the data, we use the relative  $L^2$  error:  $\frac{\|u-\tilde{u}\|_2}{\|u\|_2}$ , and the  $R^2$  score:

$$R^{2} := 1 - \frac{\sum_{i} \|u_{i} - \tilde{u}_{i}\|_{2}^{2}}{\sum_{i} \|u_{i} - \operatorname{mean}(u_{i})\|_{2}^{2}}$$

where u is the target,  $\tilde{u}$  is the model's prediction, and i is the index for sample.

A valid generated expression is considered as the one with true mathematical meanings (i.e. can be decoded into an equation) and with (relative) error less than 100%. The percentage of valid expressions are reported and the symbolic error is computed by inputting randomized-coefficient polynomials of the form  $P(x,t) = (c_0 + c_1t + c_2t^2)(c_3 + c_4x + c_5x^2 + c_6x^3 + c_7x^4)$  into the learned PDE and the true PDE then taking the relative  $L^2$  error between them. The degree of the polynomials were chosen to avoid the true PDEs from being identically zero. The Time-Series error is the relative  $L^2$  error using the prediction generated using the (particle filtered) refined PDE and initial conditions in the input data.

#### A.3 Training

The models are trained using the AdamW optimizer with batch size of 512 for 30 epochs, where each epoch is 2K steps. The learning rate scheduler is set to have 10% warmup and a cosine scheduler. We use a learning rate of  $10^{-4}$  and weight decay of  $10^{-4}$ . On a single NVIDIA GeForce RTX 4090 GPUs with 24 GB memory, the training takes about 3.0 hours with PROSE tree and 11.5 hours using SymPy tree.