TIME-SERIES FORECASTING AND REFINEMENT WITHIN A MULTIMODAL PDE FOUNDATION MODEL

Derek Jollie,[1] Jingmin Sun,[2] Zecheng Zhang,[3] & Hayden Schaeffer[4,*]

[1]*Department of Mathematics, Montana State University, Bozeman, Montana 59717, USA*

[2]*Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, USA*

[3]*Department of Mathematics, Florida State University, Tallahassee, Florida 32304, USA*

[4]*Department of Mathematics, University of California, Los Angeles, California 90095, USA*

*Address all correspondence to: Hayden Schaeffer, Department of Mathematics, University of California, Los Angeles, CA 90095, USA, E-mail: hayden@math.ucla.edu

Symbolic encoding has been used in multioperator learning (MOL) as a way to embed additional information for distinct time-series data. For spatiotemporal systems described by time-dependent partial differential equations (PDEs), the equation itself provides an additional modality to identify the system. The utilization of symbolic expressions alongside 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.

KEY WORDS: *multimodality, PDE foundation modeling, operator learning, symbolic regression, Bayesian filtering*

1. INTRODUCTION

Operator learning, initially developed as an application of a universal approximation property in Chen and Chen (1993, 1995), 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

The code is available at: https://github.com/JingminSun/prose_v1.

Submitted: 12/8/24; Accepted: 2/10/25; Online: 2/26/25
https://doi.10.1615/JMachLearnModelComput.2025057618
2689–3967/25/$35.00

made operator learning a crucial tool in computational science and scientific machine learning (Karniadakis et al., 2021; Li et al., 2020a,b; Lu et al., 2021; Raissi et al., 2019; Schaeffer, 2017; Schaeffer et al., 2013, 2018; Schaeffer and McCalla, 2017; Schaeffer and Osher, 2013; Zhang and Schaeffer, 2019). Many deep neural operators (DNOs) (Li et al., 2020a, 2022; Lin et al., 2023b; Lu et al., 2021, 2022; Wen et al., 2022; Zhang et al., 2023, 2024) have been developed and show effectiveness in solving different types of problems relating to time-dependent prediction. For example, Lu et al. (2021) introduced the deep operator network (DeepONet) for approximating the solution map for ordinary differential equations (ODEs) and PDEs. Efendiev et al. (2022) and Lin et al. (2023a) 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 interval, multioperator learning (MOL) (Liu et al., 2023, 2024; Sun et al., 2024a,b; Yang et al., 2023a,b; Yang and Osher, 2024; Zhang, 2024) 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 (Liu et al., 2023, 2024; Sun et al., 2024a,b) and ICON (Yang et al., 2022, 2023a; Yang and Osher, 2024). 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 (Schaeffer, 2017; Schaeffer et al., 2013, 2018; Schaeffer and McCalla, 2017; Sun et al., 2020; Zhang and Schaeffer, 2019) 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 (Sun et al., 2024a,b). Additionally, PROSE's symbolic encoding can be fine-tuned for downstream tasks and enables zero-shot prediction for new operators (Sun et al., 2024b). 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 (Meurer et al., 2017) 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 (Andrieu et al., 2010; Douc and Cappé, 2005; Doucet et al., 2000; Doucet and Johansen, 2011; Lin et al., 2022) 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 Fig. 1.

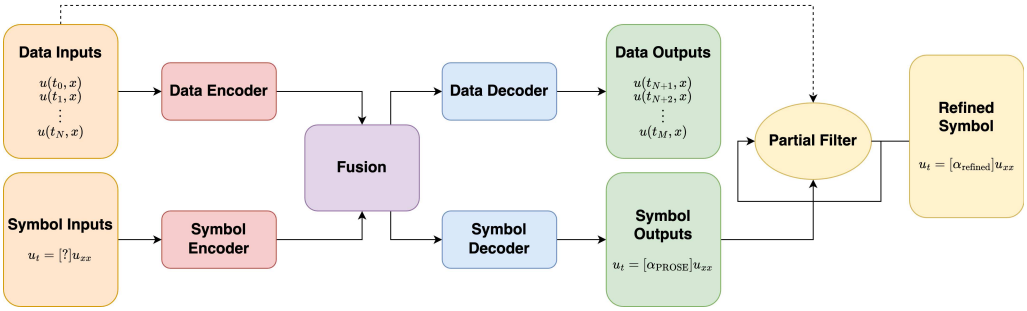


FIG. 1: PROSE PDE foundation model with particle filtering

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.

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, $i = 1, \ldots, N_{op}$, and θ 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 × 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. Notably, we use the sign, mantissa, and exponent to represent numbers (Charton, 2022; d'Ascoli et al., 2022; Kamienny et al., 2022; Liu et al., 2023). Specifically, each number is represented with three components: sign, mantissa, and exponent, which are treated as individual tokens with trainable embeddings. For example, if mantissa length is chosen to be 3, then $2.6 = +1 \times 260 \times 10^{-2}$ is represented as `[+ 260 E−2]`; see also Liu et al. (2023) for additional examples.

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 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.

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 α_0, i.e., $g_0(\alpha_0) = \text{Unif.}(0.9\alpha_0, 1.1\alpha_0)$. The parameter refinement update rule for α_k is defined as

$$\alpha_{k|k-1} = \alpha_{k-1} + \nu,$$

where $\nu \sim \mathcal{N}(0, \epsilon_p^2)$ is zero-mean Gaussian noise. Using the Chapman–Kolmogorov equation, we compute the prior belief distribution:

$$g_{k|k-1}(\alpha_{k|k-1}) = \int_{-\infty}^{\infty} p(\alpha_{k|k-1}|\alpha_{k-1}) g_{k-1}(\alpha_{k-1}) d\alpha_{k-1}. \tag{1}$$

By the update rule that $\alpha_{k|k-1} = \alpha_{k-1} + \nu$, we can infer that

$$p(\alpha_{k|k-1}|\alpha_{k-1}) = f_\nu(\alpha_{k|k-1} - \alpha_{k-1}) = f_\nu(\nu),$$

where f_ν is the pdf of the distribution of ν. Hence, we can write Eq. (1) as

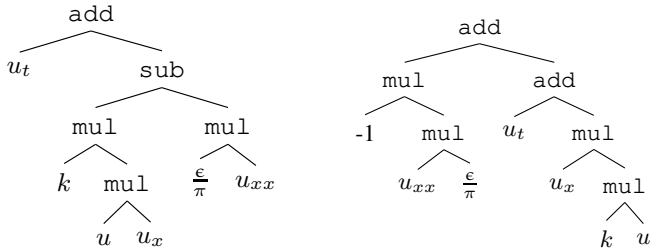


FIG. 2: PROSE tree examples: the left tree is an example of a manually standardized PROSE tree for the viscous Burgers' equation $u_t + kuu_x = (\epsilon/\pi)u_{xx}$. In the experiments, to generate the randomized trees (or a tree encountered in testing), we randomly switch the order of any branch of 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.

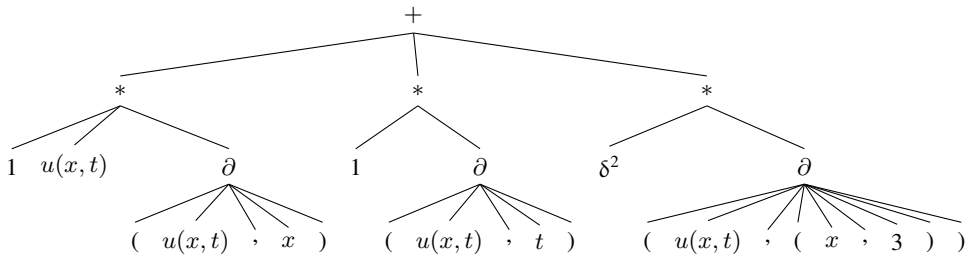


FIG. 3: SymPy Tree Example: KdV equation $uu_x + u_t + \delta^2 u_{xxx} = 0$. Here, $\partial(u(x,t),(x,3))$ is used in the tree structure to embed the term u_{xxx} and similarly. Other derivatives are written using this notation.

$$g_{k|k-1}(\alpha_{k|k-1}) = \int_{-\infty}^{\infty} f_\nu(\nu) g_{k-1}(\alpha_{k-1}) d\alpha_{k-1}. \tag{2}$$

Bayes' theorem then gives the posterior belief with normal coefficient $\eta = 1/p(u(t_k,\cdot))$:

$$g_k(\alpha_k) = p(\alpha_k|u(t_k,\cdot)) = \eta\, p(u(t_k,\cdot)|\alpha_k) g_{k|k-1}(\alpha_k). \tag{3}$$

This process is known as the Bayesian filtering (Chen, 2003), and in practice, we implement it using a SMC particle filter simulation (Andrieu et al., 2010; Doucet et al., 2000; Doucet and Johansen, 2011; Gustafsson, 2010; Lin et al., 2022). The details appear in Section 2.1 and Fig. 4.

2.1 Particle Filter

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

$$u(\cdot,t_k) = H(\alpha_k, u(\cdot,t_{k-1})) + \sigma, \tag{4}$$

where H is a (deterministic) numerical scheme for solving the PDE and σ is the observation noise. In our case σ follows a probability density function $f_\sigma(\sigma)$ and is sampled from a zero-mean Gaussian distribution with variance ϵ_o^2, i.e., $\sigma \sim \mathcal{N}(0,\epsilon_o^2)$. Hence the likelihood of observing $u(\cdot,t_k)$ given α_k is

$$p\big(u(t_k,\cdot)|\alpha_k\big) = f_\sigma\big(u(\cdot,t_k) - H(\alpha_k, u(\cdot,t_{k-1}))\big) = f_\sigma(\sigma), \tag{5}$$

and inserting it into Eq. (3) yields

$$g_k(\alpha_k) = \eta f_\sigma(\sigma) g_{k|k-1}(\alpha_k). \tag{6}$$

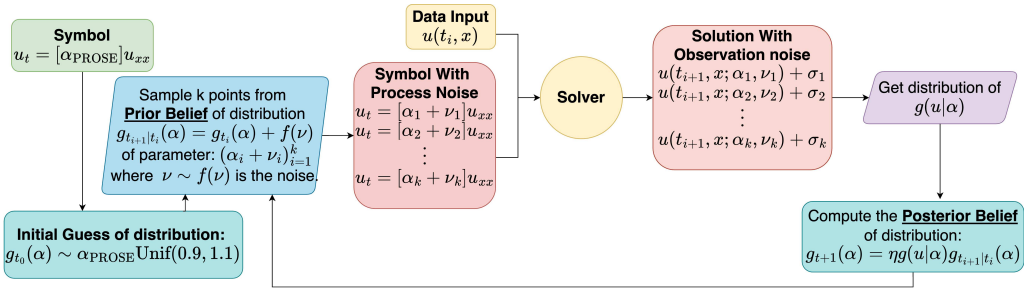


FIG. 4: Particle filter module: a discretized version of the Bayesian filter process

To compute this, we utilize a SMC particle method where we generate M particles $\{\alpha_{k-1}^{(i)}\}_{i=1}^{M}$ from the distribution $g_{k-1}(\alpha_{k-1})$ and approximate Eq. (2) by

$$g_{k|k-1}(\alpha_{k|k-1}) \approx \frac{1}{M}\sum_{i=1}^{M}\delta\big(\alpha_{k-1}^{(i)} + \nu_i\big) = \frac{1}{M}\sum_{i=1}^{M}\delta\big(\alpha_{k|k-1}^{(i)}\big), \tag{7}$$

where $\delta(\cdot)$ represents the Dirac delta function, which ensures that the empirical distribution is represented as a sum of point masses at the predicted particle locations. Each particle evolves according to the update rule:

$$\alpha_{k|k-1}^{(i)} = \alpha_{k-1}^{(i)} + \nu_i,$$

where ν_i is sampled independently from a zero-mean Gaussian distribution with variance ϵ_p^2.

Next, we calculate importance weights (Geweke, 1989), which are given by

$$w_i \propto \frac{g_k\big(\alpha_k^{(i)}\big)}{g_{k|k-1}\big(\alpha_k^{(i)}\big)} = f_\sigma(\sigma), \qquad p_i = \frac{w_i}{w_0}, \tag{8}$$

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 α_k:

$$\begin{aligned} G_k(\alpha_k) &= \int_{-\infty}^{\alpha_k} g_k(\zeta)d\zeta \\ &= \int_{-\infty}^{\alpha_k} \eta f_\sigma(\sigma) g_{k|k-1}(\zeta)d\zeta \\ &\approx \frac{1}{\sum_{i=1}^{M} f_\sigma(\sigma)}\sum_{i=1}^{M} f_\sigma(\sigma)\mathbb{I}\big(\alpha_{k|k-1}^{(i)} \le \alpha_k\big) \qquad (9) \\ &= \sum_{i=1}^{M} p_i\, \mathbb{I}\big(\alpha_{k|k-1}^{(i)} \le \alpha_k\big), \qquad (10) \end{aligned}$$

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

In this work, the particle-based refinement process uses $M = 500$ particles and uses ten refinement steps. At the last step, we output the mean of the distribution $G_{10}(\alpha_{10})$ as the refined parameter, i.e., $\alpha_{\text{refined}} = (1/M)\sum_{i=1}^{M}\alpha_{10}^{(i)}$. The process noise is modeled as a normal distribution with $\epsilon_p = 10^{-5}$. 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., $\epsilon_o = 0.05\|u(\cdot, t=0)\|_2$.

3. EXPERIMENT SETUP

3.1 Dataset

The dataset utilizes the conservation laws from Sun et al. (2024a). To summarize, it consists of six 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 rescale 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.

3.2 Evaluation Metrics

Since we use two modalities, we utilize four evaluation metrics from Sun et al. (2024a). For metrics on the data, we use the relative L^2 error: $\|u - \tilde{u}\|_2 / \|u\|_2$, and the R^2 score:

$$R^2 := 1 - \frac{\sum\limits_i \|u_i - \tilde{u}_i\|_2^2}{\sum\limits_i \|u_i - \text{mean}(u_i)\|_2^2},$$

where u is the target, $\tilde{u}$ is the model's prediction, and i is the index for the 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 percentages of valid expressions are reported and the symbolic error is computed by inputting randomized-coefficient polynomials of the form

$$P(x, t) = (c_0 + c_1 t + c_2 t^2)(c_3 + c_4 x + c_5 x^2 + c_6 x^3 + c_7 x^4),$$

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.

3.3 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 a weight decay of 10^{-4}. On a single NVIDIA GeForce RTX 4090 GPUs with 24 GB memory, the training takes about 3.0 hr with the PROSE tree and 11.5 hr using the SymPy tree.

4. 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 Sun et al. (2024a); (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 Fig. 2.

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%. Note that when comparing between the noisy cases, one advantage of the SymPy tree is that the added consistency leads to a higher valid fraction, likely indicating more robust knowledge extraction. Moreover, the valid fraction, representing the ratio of mathematically valid predicted PDEs, correlates with the symbolic error: lower prediction errors correspond to higher valid fractions. Notably, the proposed SymPy Tree substantially enhances the validity of the generated symbols.

The examples for which the symbolic expressions are invalid tend to be a consequence of issues with resolving numerical values. For example, a generated PROSE tree with noisy inputs may contain the term "$\cos(N915x)$," where $N915$ stands for the mantissa part of a number, but with the sign and exponent part missing. Consequently, we cannot recover a floating number and this results in an invalid prediction. For generated Sympy trees with noisy inputs, invalid expressions tend to have terms such as "$E0E - u\cos(x)$," in which the invalid term "$E0E$" will not affect other terms. This also explains the reason SymPy tree yields a higher valid fraction.

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 Figs. 5 and 6.

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 Section 3.2 for details. PROSE tree* uses manual formatting and thus is not a direct comparison

Noise	Testing Tree Structure	Relative L^2 Error	L^2 Score	Symbolic Error	Valid Fraction
Noise-Free	PROSE Tree*	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%

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 Section 3.2). (I)CL: (Inviscid) conservation law

Type of equation	Expression	Symbolic Error		Time-Series Error	
		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%

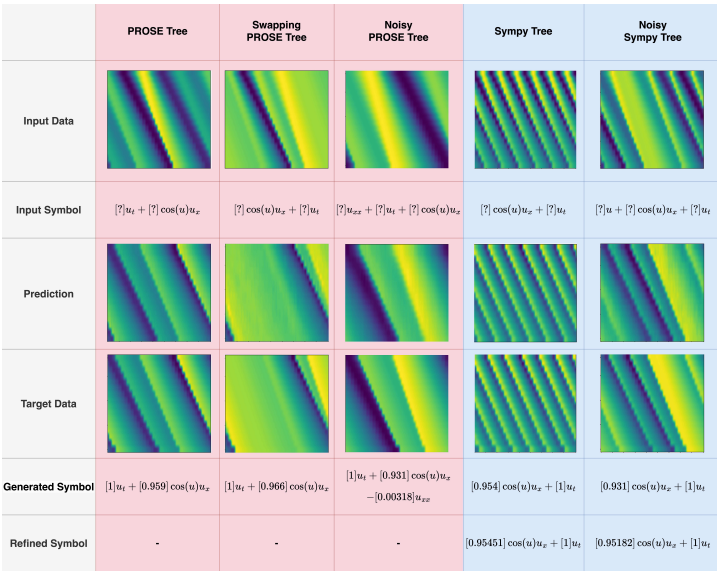


FIG. 5: Various examples of the symbolic modality for inviscid conservation law with sine flux. Target equation: $u_t + 0.955\cos(u)u_x = 0$. For PROSE tree, the model is trained for the order $[?]u_t + [?]\cos(u)u_x$, and for SymPy tree, the input expression is automatically uniformed into $[?]\cos(u)u_x + [?]u_t$. The generated symbols use three 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.

5. ABLATION STUDY

In this section, we explore the behavior of particle filtering under varying numbers of steps and analyze its sensitivity to the observation and process noise.

5.1 Filtering Steps

As presented in Fig. 7, when the number of filtering steps increases, the model's performance improves, as expected. Notably, with only a single filtering step, the error may even increase due to insufficient filtering. As the number of steps increases further, the model stabilizes, and

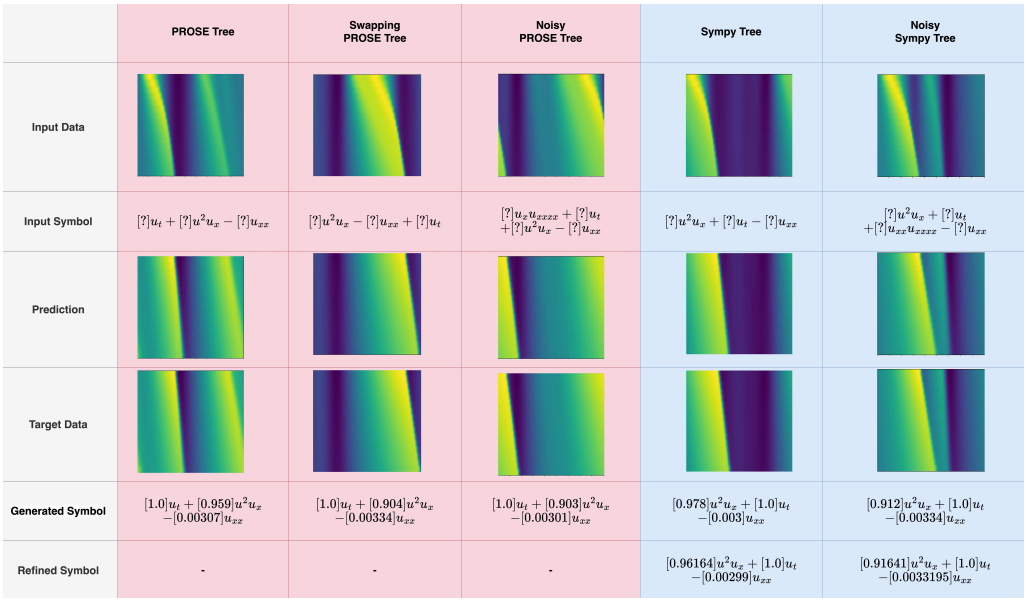
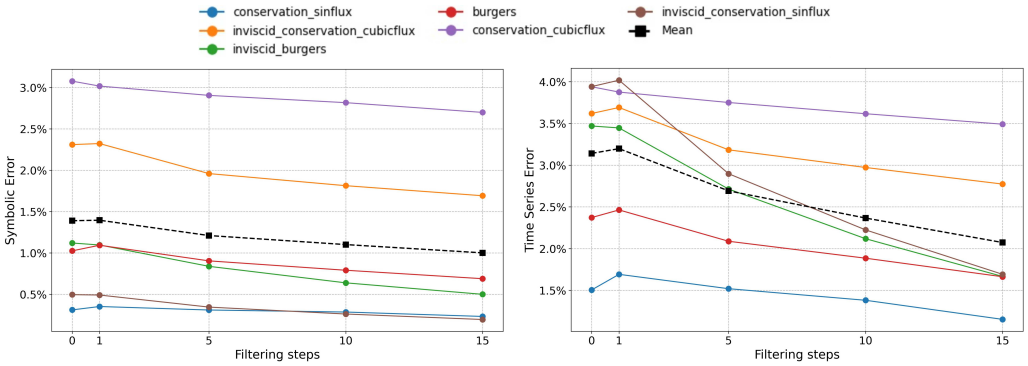


FIG. 6: Various examples of the symbolic modality for viscous conservation law with cubic flux. Target equation: $u_t + 0.936u^2u_x - 0.0289u_{xx} = 0$. For PROSE tree, the model is trained for the order $[?]u_t + [?]u^2u_x - [?]u_{xx}$, and for SymPy tree, the input expression is automatically uniformed into $[?]\cos(u)u_x + [?]u_t - [?]u_{xx}$. The generated symbols use three 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.



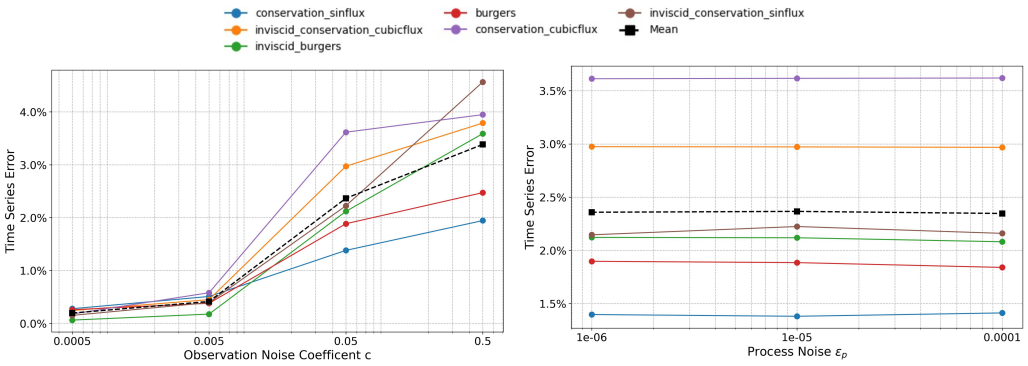
(a) (b)

FIG. 7: Trends in symbolic and time-series errors with varying filtering steps. (a) Symbolic error as a function of filtering steps. (b) Time-series error as a function of filtering steps.

the error reduction becomes more consistent. To balance performance gains with computational cost, we set the number of filtering steps to 10 in our main results.

5.2 Observation and Process Noises

The standard deviation of the observation noise ϵ_o and the process noise ϵ_p as defined in Section 2 play an important role in the filtering process. As shown in Fig. 8, the errors are influenced



(a) (b)

FIG. 8: Trends in time-series errors with varying noises after 10-step filtering. (a) Time-series error as a function of standard deviation of observation noise coefficients, $\epsilon_o = c\|u(\cdot, t = 0)\|_2$. (b) Time-series error as a function of standard deviation of process noise ϵ_p.

more by changes to the observation noise, but not the process noise. In our main results, we set the coefficients of observation noise to $c = 0.05$, and set the standard deviation of process noise to 10^{-5}.

6. CONCLUSIONS

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 advanced refinement techniques to address limitations of the current particle filter framework, such as degeneracy issues that reduce the effective particle size. Exploring applications to larger, higher-dimensional datasets is important for evaluating scalability.

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

Andrieu, C., Doucet, A., and Holenstein, R., Particle Markov Chain Monte Carlo Methods, *J. R. Stat. Soc. Ser. B: Stat. Methodol.*, vol. **72**, no. 3, pp. 269–342, 2010.

Charton, F., Linear Algebra with Transformers, arXiv preprint arXiv:2112.01898, 2022.

Chen, T. and Chen, H., Approximations of Continuous Functionals by Neural Networks with Application to Dynamic Systems, *IEEE Trans. Neural Networks*, vol. **4**, no. 6, pp. 910–918, 1993.

Chen, T. and Chen, H., Universal Approximation to Nonlinear Operators by Neural Networks with Arbitrary Activation Functions and Its Application to Dynamical Systems, *IEEE Trans. Neural Networks*, vol. **6**, no. 4, pp. 911–917, 1995.

Chen, Z., Bayesian Filtering: From Kalman Filters to Particle Filters, and Beyond, *Statistics*, vol. **182**, no. 1, pp. 1–69, 2003.

d'Ascoli, S., Kamienny, P.A., Lample, G., and Charton, F., Deep Symbolic Regression for Recurrent Sequences, arXiv preprint arXiv:2201.04600, 2022.

Douc, R. and Cappé, O., Comparison of Resampling Schemes for Particle Filtering, in *ISPA 2005, Proc. of the 4th Int. Symp. on Image and Signal Processing and Analysis*, Zagreb, Croatia, pp. 64–69, 2005.

Doucet, A., Godsill, S., and Andrieu, C., On Sequential Monte Carlo Sampling Methods for Bayesian Filtering, *Stat. Comput.*, vol. **10**, pp. 197–208, 2000.

Doucet, A. and Johansen, A.M., A Tutorial on Particle Filtering and Smoothing: Fifteen Years Later, in *The Oxford Handbook of Nonlinear Filtering*, D. Crisan and B. Rozovskii, Eds., Oxford: Oxford University Press, 2011.

Efendiev, Y., Leung, W.T., Lin, G., and Zhang, Z., Efficient Hybrid Explicit-Implicit Learning for Multiscale Problems, *J. Comput. Phys.*, p. 111326, 2022.

Geweke, J., Bayesian Inference in Econometric Models Using Monte Carlo Integration, Econometrica: *J. Econometric. Soc.*, vol. **57**, no. 6, pp. 1317–1339, 1989.

Gustafsson, F., Particle Filter Theory and Practice with Positioning Applications, *IEEE Aerospace Electron. Syst. Mag.*, vol. **25**, no. 7, pp. 53–82, 2010.

Kamienny, P.A., d'Ascoli, S., Lample, G., and Charton, F., End-to-End Symbolic Regression with Transformers, in *Advances in Neural Information Processing Systems*, A.H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., Cambridge, MA: MIT Press, 2022.

Karniadakis, G.E., Kevrekidis, I.G., Lu, L., Perdikaris, P., Wang, S., and Yang, L., Physics-Informed Machine Learning, *Nat. Rev. Phys.*, vol. **3**, no. 6, pp. 422–440, 2021.

Li, Z., Huang, D.Z., Liu, B., and Anandkumar, A., Fourier Neural Operator with Learned Deformations for PDEs on General Geometries, arXiv preprint arXiv:2207.05209, 2022.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A., Fourier Neural Operator for Parametric Partial Differential Equations, arXiv preprint arXiv:2010.08895, 2020a.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A., Neural Operator: Graph Kernel Network for Partial Differential Equations, arXiv preprint arXiv:2003.03485, 2020b.

Lin, G., Moya, C., and Zhang, Z., B-DeepONet: An Enhanced Bayesian DeepONet for Solving Noisy Parametric PDEs Using Accelerated Replica Exchange SGLD, *J. Comput. Phys.*, vol. **473**, p. 111713, 2023a.

Lin, G., Moya, C., and Zhang, Z., Learning the Dynamical Response of Nonlinear Non-Autonomous Dynamical Systems with Deep Operator Neural Networks, *Eng. Appl. Artif. Intell.*, vol. **125**, p. 106689, 2023b.

Lin, G., Zhang, Z., and Zhang, Z., Theoretical and Numerical Studies of Inverse Source Problem for the Linear Parabolic Equation with Sparse Boundary Measurements, *Inverse Probl.*, vol. **38**, no. 12, p. 125007, 2022.

Liu, Y., Sun, J., He, X., Pinney, G., Zhang, Z., and Schaeffer, H., PROSE-FD: A Multimodal PDE Foundation Model for Learning Multiple Operators for Forecasting Fluid Dynamics, arXiv preprint arXiv:2409.09811, 2024.

Liu, Y., Zhang, Z., and Schaeffer, H., Prose: Predicting Operators and Symbolic Expressions Using Multimodal Transformers, arXiv preprint arXiv:2309.16816, 2023.

Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G.E., Learning Nonlinear Operators via DeepONet Based on the Universal Approximation Theorem of Operators, *Nat. Mach. Intell.*, vol. **3**, no. 3, pp. 218–229, 2021.

Lu, L., Meng, X., Cai, S., Mao, Z., Goswami, S., Zhang, Z., and Karniadakis, G.E., A Comprehensive and Fair Comparison of Two Neural Operators (with Practical Extensions) Based on Fair Data, *Comput. Methods Appl. Mech. Eng.*, vol. **393**, p. 114778, 2022.

Meurer, A., Smith, C.P., Paprocki, M., Čertík, O., Kirpichev, S.B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J.K., and Singh, S., SymPy: Symbolic Computing in Python, *PeerJ Comput. Sci.*, vol. **3**, p. e103, 2017.

Raissi, M., Perdikaris, P., and Karniadakis, G.E., Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations, *J. Comput. Phys.*, vol. **378**, pp. 686–707, 2019.

Schaeffer, H., Learning Partial Differential Equations via Data Discovery and Sparse Optimization, *Proc. R. Soc. A: Math. Phys. Eng. Sci.*, vol. **473**, no. 2197, p. 20160446, 2017.

Schaeffer, H., Caflisch, R., Hauck, C.D., and Osher, S., Sparse Dynamics for Partial Differential Equations, *Proc. Natl. Acad. Sci.*, vol. **110**, no. 17, pp. 6634–6639, 2013.

Schaeffer, H. and McCalla, S.G., Sparse Model Selection via Integral Terms, *Phys. Rev. E*, vol. **96**, no. 2, p. 023302, 2017.

Schaeffer, H. and Osher, S., A Low Patch-Rank Interpretation of Texture, *SIAM J. Imag. Sci.*, vol. **6**, no. 1, pp. 226–262, 2013.

Schaeffer, H., Tran, G., and Ward, R., Extracting Sparse High-Dimensional Dynamics from Limited Data, *SIAM J. Appl. Math.*, vol. **78**, no. 6, pp. 3279–3295, 2018.

Sun, J., Liu, Y., Zhang, Z., and Schaeffer, H., Towards a Foundation Model for Partial Differential Equations: Multi-Operator Learning and Extrapolation, arXiv preprint arXiv:2404.12355, 2024a.

Sun, J., Zhang, Z., and Schaeffer, H., LeMON: Learning to Learn Multi-Operator Networks, arXiv preprint arXiv:2408.16168, 2024b.

Sun, Y., Zhang, L., and Schaeffer, H., NeuPDE: Neural Network Based Ordinary and Partial Differential Equations for Modeling Time-Dependent Data, *Math. Sci. Mach. Learn.*, vol. **107**, pp. 352–372, 2020.

Wen, G., Li, Z., Azizzadenesheli, K., Anandkumar, A., and Benson, S.M., U-FNO—An Enhanced Fourier Neural Operator-Based Deep-Learning Model for Multiphase Flow, *Adv. Water Resour.*, vol. **163**, p. 104180, 2022.

Yang, L., Liu, S., Meng, T., and Osher, S.J., In-Context Operator Learning with Data Prompts for Differential Equation Problems, *Proc. Natl. Acad. Sci.*, vol. **120**, no. 39, p. e2310142120, 2023a.

Yang, L., Meng, T., Liu, S., and Osher, S.J., Prompting In-Context Operator Learning with Sensor Data, Equations, and Natural Language, arXiv preprint arXiv:2308.05061, 2023b.

Yang, L. and Osher, S.J., PDE Generalization of In-Context Operator Networks: A Study on 1D Scalar Nonlinear Conservation Laws, arXiv preprint arXiv:2401.07364, 2024.

Yang, Y., Kissas, G., and Perdikaris, P., Scalable Uncertainty Quantification for Deep Operator Networks Using Randomized Priors, *Comput. Methods Appl. Mech. Eng.*, vol. **399**, p. 115399, 2022.

Zhang, L. and Schaeffer, H., On the Convergence of the SINDy Algorithm, *Multiscale Model. Simul.*, vol. **17**, no. 3, pp. 948–972, 2019.

Zhang, Z., MODNO: Multi-Operator Learning with Distributed Neural Operators, *Comput. Methods Appl. Mech. Eng.*, vol. **431**, p. 117229, 2024.

Zhang, Z., Moya, C., Lu, L., Lin, G., and Schaeffer, H., D2NO: Efficient Handling of Heterogeneous Input Function Spaces with Distributed Deep Neural Operators, *Comput. Methods Appl. Mech. Eng.*, vol. **428**, p. 117084, 2024.

Zhang, Z., Wing Tat, L., and Schaeffer, H., BelNet: Basis Enhanced Learning, a Mesh-Free Neural Operator, *Proc. R. Soc. A*, vol. **479**, no. 2276, p. 20230043, 2023.