

# Towards a Foundation Model for Partial Differential Equations: Multi-Operator Learning and Extrapolation

Jingmin Sun\*    Yuxuan Liu<sup>†</sup>    Zecheng Zhang<sup>‡</sup>    Hayden Schaeffer<sup>†</sup>

## Abstract

Foundation models, such as large language models, have demonstrated success in addressing various language and image processing tasks. In this work, we introduce a multi-modal foundation model for scientific problems, named PROSE-PDE. Our model, designed for bi-modality to bi-modality learning, is a multi-operator learning approach which can predict future states of spatiotemporal systems while concurrently learning the underlying governing equations of the physical system. Specifically, we focus on multi-operator learning by training distinct one-dimensional time-dependent nonlinear constant coefficient partial differential equations, with potential applications to many physical applications including physics, geology, and biology. More importantly, we provide three extrapolation studies to demonstrate that PROSE-PDE can generalize physical features through the robust training of multiple operators and that the proposed model can extrapolate to predict PDE solutions whose models or data were unseen during the training. Furthermore, we show through systematic numerical experiments that the utilization of the symbolic modality in our model effectively resolves the well-posedness problems with training multiple operators and thus enhances our model’s predictive capabilities.

## 1 Introduction

Partial differential equations (PDEs) are fundamental models for describing and understanding complex spatio-temporal processes in the physical and computational sciences. They provide one of the most important techniques for bridging experimental observations, physical principles, and mathematical properties. PDEs are effective in describing, analyzing, and predicting a wide array of real-world phenomena, including highly nonlinear, chaotic, and/or multi-scale physics. Scientific computing (SC) problems concerning PDE center around simulating a specific equation or class of equations given a set of parameters, initial states, boundary conditions, etc. The objective is often to match experimental findings with formal models, forecast state variables, understand parametric dependencies, or obtain physical parameters.

When formulated as a machine learning task, solving PDEs amounts to approximating a mapping between input parametric functions and output solutions, equations, or more broadly, output information. In this work, we introduce the PROSE-PDE model, designed to be a foundation model for solving both forward and inverse PDE problems. The model is trained across various classes of

---

\*Department of Mathematical Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA.

<sup>†</sup>Department of Mathematics, University of California, Los Angeles, Los Angeles, CA 90095, USA.

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

time-dependent PDEs, including nonlinear diffusive, dispersive, conservation laws, wave equations, and others. The goal is to produce a neural network model that is able to generalize solutions based on different parametrized conditions and to extrapolate important physical phenomena between different governing systems. This is a key step towards a foundation model for scientific applications.

*Foundation models* are deep learning models trained on large datasets often comprising of billions of learnable parameters in order to serve as base models readily applicable across a broad spectrum of applications [3]. They have significantly revolutionized natural language processing and the broader AI landscape through models such as BERT [10], GPT [4, 44, 45], DALL-E [46, 47], Stable Diffusion [48], LLAMA [57, 58], Claude, and others. Foundation models are general-purpose, allowing fine-tuning of parameters on additional datasets for new downstream tasks. However, to ensure their efficacy across a wide array of conditions, foundation models necessitate a significant amount of training data (trillions of tokens [57]) and computational resources.

While generative AI has seen unprecedented success in text-based tasks and image generation, its application to SC problems remains limited. There are several key differences that make SC problems challenging for current large language models (LLMs). Firstly, accuracy and precision are crucial in SC applications. For example, a small numerical error in a coefficient of a (trained) governing equation can lead to catastrophic errors in downstream computing tasks, especially for chaotic, high-contrast, or multi-scale systems where errors accumulate rapidly. This also holds for other SC tasks such as forecasting state variables or optimizing structural parameters. Secondly, SC problems typically yield unique solutions, e.g., there is only one solution to a well-posed PDE given a fixed set of conditions, whereas a sentence can convey a similar meaning with a variety of word choices. Additionally, while LLMs show some forms of reasoning abilities or emergent behaviors, they are not yet capable of numerical or mathematical reasoning, including understanding relations, ordering, properties, and symmetries [17, 79]. Though recent efforts have focused on chain-of-thought reasoning prompts to elucidate step-by-step reasoning processes, in [60] it was shown that a model could give a plausible argument that is consistent with the predicted answer but is an “unfaithful explanation of the model’s decision procedure.” For the trustworthiness of LLMs in the scientific domain, being able to provide reasoning or certified guarantees are necessary. Lastly, SC works in the data-scarce regime, as experimental data often requires time to acquire and is far less available compared to text or imaging databases. Thus, numerical simulations and heterogeneous data sources play a stronger role in training and testing foundation models for scientific applications.

## 1.1 Main Contributions

This work introduces a multi-modal neural network approach, PROSE-PDE, for predicting solutions of 1D time-dependent PDE systems and for generating the underlying equations. The workflow of PROSE-PDE is illustrated in Figure 1. The main contributions and novelties are summarized below.

- PROSE-PDE is the first multi-modal transformer-based approach that encodes and decodes both numerical and symbolic datatypes (i.e. forward and inverse problems for multiple classes of PDEs). PROSE-PDE addresses the challenging problem of multi-operator learning.
- We propose an approach that can generalize to new model/physical parameter values not encountered during training, to unseen timestamps or further into the future, to new initial condition distributions, to unseen physical systems, and to new physical features, all without

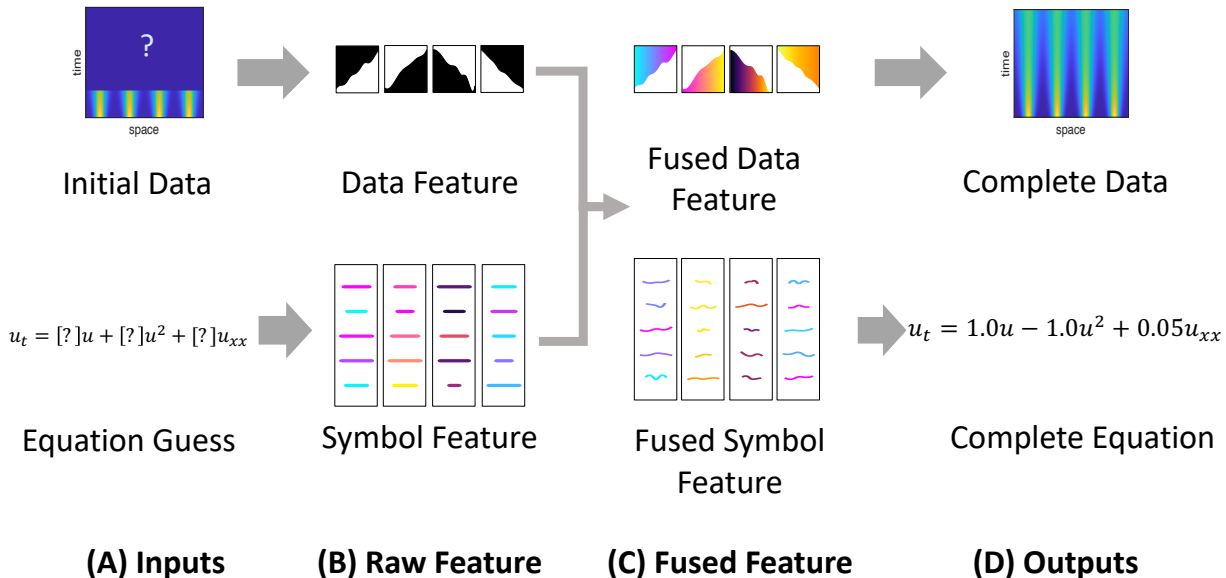


Figure 1: **PROSE-PDE Workflow Illustration:** The inputs are the initial data (short-time observations) and a guess of the partial differential equation itself. The inputs are mapped into raw features (modality-specific) and then fused together. The fusion process couples cross-modality information. The decoders output a prediction of the data (as an operator) and write a complete mathematically valid equation.

fine-tuning. Three detailed studies, which include thirteen experimental settings, demonstrate the model’s extrapolation capabilities.

- We conduct two ablation experiments varying (1) the input length in time and (2) adjusting the weighting between the losses (data and symbolic), in order to examine the contribution of each of the two symbolic modalities (input and output) to the learning process. The results demonstrate the benefit of multi-modal information and the model’s consistency under different training settings.

## 2 Overview

In this section, we introduce operator learning, provide an overview of the main technical aspects of the proposed model, and discuss related works.

### 2.1 Multi-Operator Learning (MOL)

Let  $G : U \rightarrow V$  be an operator, where  $U$  and  $V$  represent function spaces. Single operator learning (SOL) aims to construct one neural operator  $G_\theta$  parameterized by  $\theta$  to approximate the operator  $G$ . In [6, 7], the authors proved the first universal approximation theorem for nonlinear operators, which has led to several recent approaches in operating learning. For example, the popular Deep Operator Neural Network (DON) [20, 31, 32, 35, 36] and its variants [74, 75, 77] approximate the operator by learning the function bases and have the benefit of not requiring a fixed discretization for the output function. Fourier Neural Network (FNO) [26, 27, 28, 29, 63] utilizes Fourier layers

within the network architecture and thus are invariant to the input discretization. These advancements have led to successes in applying neural operators to various real-life engineering challenges; for example, those in climate [19, 42], earth structural [78], physics and astronomy [11, 38, 39], biology [71], power systems [32], optimization and UQ [31, 41], multi-fidelity modeling [16, 23, 37, 76], etc.

SOL has been used in solving mathematical and scientific computing problems. For example, SOL can construct the solution operator of PDEs and learn the mapping from the initial condition of the system to the solution. Specifically, let us denote a parametrized PDE system  $\mathcal{S}$  as,

$$\begin{aligned}\mathcal{L}(u(x, t; q)) &= 0, \quad x \in \Omega, \\ \mathcal{B}(u(x, t; q)) &= 0, \quad x \in \partial\Omega, \\ u(x, 0; q) &= \mathcal{G}(x; q), \quad x \in \Omega, \quad q \sim \mathcal{D},\end{aligned}$$

where  $\mathcal{L}$  and  $\mathcal{B}$  denote the governing equations and boundary conditions. The initial condition  $\mathcal{G}$  is a generating function,  $q$  denotes the parameters that determine the initial conditions, and  $\mathcal{D}$  is the distribution. Thus, one task in SOL is to train the mapping from  $\mathcal{G}(x; q)$  to  $u(x, t; q)$ , i.e. learn  $G \approx G_\theta$ .

A central goal in scientific machine learning is to develop methods that are able to extrapolate, i.e. to solve tasks beyond those encountered during training. While a trained SOL model may effectively address a specific operator and task, it could encounter difficulties when presented with new tasks. For instance, if  $G_\theta$  learns an approximation to the solution operator  $G^{(0)}$  for system  $\mathcal{S}^{(0)}$ , it may struggle to handle a new task represented by a different operator  $G^{(1)}$  associated with a new system  $\mathcal{S}^{(1)}$ . Thus to address this challenge, we develop a multi-operator learning (MOL) approach, which entails training a single foundation model to learn multiple operators. Specifically, the objective is to establish an umbrella mapping that can map many distinct encoded operators (with their corresponding input functions) to an accurate approximation of their output function.

Several MOL methods [33, 40, 52, 67, 68, 69, 70, 73] have recently been proposed. Most of these methods provide a label (either implicitly or explicitly) to the inputs, signaling to the model which among the many operators to use. The label is one approach for dealing with the issue of well-posedness, i.e. disambiguating different equations with similar initial values. The PROSE (Predicting Operators and Symbolic Expression) approach [33] is a multi-modal foundation model capable of constructing operators while simultaneously learning equations. Utilizing trainable symbols to encode operators, PROSE has demonstrated efficacy in encoding high-dimensional dynamical systems and for learning mathematical representation of the data.

The In-Context Operator Network (ICON) [67, 68, 69] uses an in-context learning approach for MOL, where existing data from the encountered system is used as implicit labels for the model. In [70], the model uses a graph transformer to process the graph representation of the equation, which may suffer from long-range dependency issues and is limited to forward problems. In [52], a pretrained LLM is used to generate equation labels for the model, and the model’s ability to reason about equations is thus dependent on how well the LLM understands mathematical and numerical values. Multiple Physics Pretraining (MPP) [40] directly learns from only the history of the system. Since no labels are used, MPP may suffer from the well-posedness issues.

While many of these methods show some ability to generalize to new system parameters outside of the training range; none have demonstrated an ability to generalize to unseen systems and extrap-



olate physical phenomena without the use of fine-tuning. As the training dataset is rarely exhaustive, this is a natural requirement for SC foundation models. In this work, we propose a multi-modal scientific foundation model, PROSE-PDE, in the context of 1D PDE problems and we show the model’s extrapolation capabilities.

## 2.2 Multi-Modal Machine Learning (MMML)

Multi-modal machine learning (MMML) trains models using data from heterogeneous sources [25, 34, 53, 56, 66] and solves multiple tasks simultaneously. For example, for visual-language reasoning [24, 53, 56], models utilize visual content, i.e. images or videos, combined with the semantics of language [56] associated with these visual elements, such as captions or descriptions. This has led to the development of models with richer information [24]. MOL tasks inherently belong to the realm of MMML tasks, specifically characterized as bi-modal tasks involving functions and operator encoding as two heterogeneous inputs.

The PROSE-PDE model takes a further step: it also generates a governing equation of the system, making it a bi-modal input bi-modal output model. We present the details in Section 3. Numerous innovations contribute to the efficacy of large-scale foundational multi-modal models, with one pivotal advancement being the integration of attention and transformer structures. The attention mechanism allows for complex sequence encodings, adept at capturing sequential dependencies within data [2, 8, 62]. When processing features, this mechanism evaluates the significance of various segments within the sequence, generating distinct encodings for each segment. It then utilizes these encodings to attend to different parts of the sequence by varying the weights, fostering intra-sequence connectivity. The classic transformer architecture leverages self-attention [1, 65], enabling it to capture intricate relationships within lengthy sequential data. Conversely, cross-attention enables the model to discern connections between distinct sequences, thereby facilitating the learning of inter-modality relationships. More details are included in Appendix C.

## 2.3 Extrapolation of Physical Features (EPF)

Abstraction is a fundamental aspect of the scientific method, in particular, the pursuit of a deeper understanding of the underlying mechanisms behind observed phenomena. In this work, we validate our approach by assessing its ability to generalize across different input conditions and its capabilities of extrapolation of physical features, which we will refer to as EPF. In the setting of foundation models for scientific domain problems, extrapolation entails evaluating whether the model has learned the underlying physical laws to a sufficient extent to generalize to either new physical systems or new conditions, possibly through a transfer of fundamental rules or key features. While there is no clear definition to the extent one expects a neural network to extrapolate spatiotemporal systems, some useful capabilities include:

- Generalize to new model/physical parameter values not encountered during training,
- Predict variables at unseen timestamps or further into the future (forecasting),
- Handle new condition classes, such as changes in the smoothness of the initial state or the form of the parametric input functions,
- Generalize to new physical systems not seen in training.

In addition, a scientific foundation model should have the EPF property, since physical phenomena are often shared across systems that have similar underlying laws. For this work, we focus on the generalization of physical features in conservative systems, specifically, we show that our model has the EPF property by:

- Training the model on a dataset containing simulations with only single observed shocks, while testing it on settings with multiple shocks (extrapolating shock interactions),
- Varying the training sets with mixtures of shocks and rarefactions (with differing amounts per physical system) and asking the model to predict a shock or rarefaction on a new system (transferring physical laws to a new system).

All of these EPF tests are challenging for classical and current techniques; however, they are essential for demonstrating progress toward developing a general-purpose large-scale model for physical systems. We focus on smaller-scale problems to show that these properties hold even with (1) limited data or (2) a smaller amount of trainable parameters as compared to standard LLMs. Notably, these tests provide insights into PROSE-PDE’s capacity for abstraction or can at least measure its potential to extract underlying rules from PDE data.

### 3 Methodology

The main components of the PROSE-PDE architecture include transformers, symbolic encoding, and multi-modal inputs and outputs. We summarize some key elements in this section and provide an overview of the PROSE-PDE architecture and workflow. More architecture details are in Appendix B.

#### 3.1 Equation Encoding via Polish Notation

A central challenge in MOL is encoding operators so that the encoded representations can seamlessly adapt to new, unseen operators. We will employ symbolic encoding to address this challenge. The symbolic encoding of mathematical operators has been studied in [33] and it is also used in other mathematical problems [18, 30]. However, its extrapolation abilities to new operators have undergone limited investigation. Through numerical experiments in Section 5.1, we demonstrate that the use of symbolic encoding facilitates extrapolation. We illustrate the symbolic encoding of an equation in Figure 2. This encoding involves representing the equation in a tree structure, where nodes represent operations and leaves represent variables and constants. We then convert the tree structure into Polish notation [43]. Throughout the training process, all symbols in the Polish notation are considered trainable tokens and are updated accordingly. For further details, please refer to [5, 9, 12, 21, 33, 54].

#### 3.2 Model Overview

The PROSE-PDE architecture contains five main components: Data Encoder, Symbol Encoder, Feature Fusion, Data Decoder, and Symbol Decoder. All of these components use variations of the attention structure. In this section, we present the workflow of PROSE-PDE and illustrate how information is processed in the network (see Figure 3). Technical details of each component are

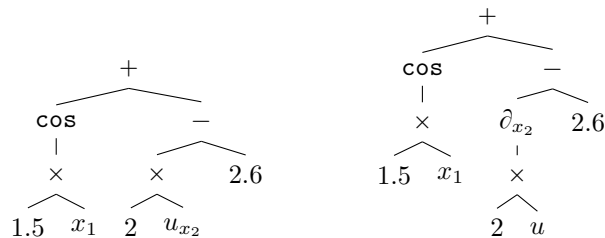


Figure 2: Two equivalent tree encodings of the example expression  $\cos(1.5x_1) + 2u_{x_2} - 2.6$ . The left tree directly uses the partial derivative symbol  $u_{x_2}$ , while the right tree uses a differential operator symbol  $\partial_{x_2}$ . We adopt the left approach for the tests in this work.

discussed in Appendix B.

There are two types of inputs to the PROSE-PDE network: the initial data sequence and the symbolic equation guesses. The two input modalities are first transformed into sequences of feature vectors, making them more suitable for subsequent tasks. A small feedforward network independently up-samples each element in the input data sequence, generating a sequence of high-dimensional feature vectors. The sequence is then processed by the Data Encoder, allowing information to flow across elements in the sequence (e.g. the Data Encoder can locate the minimum/maximum, find peaks/modes, and detect patterns in the sequence).

The symbolic equation guesses are transformed into a sequence of symbol tokens, which is processed in the same way as a sentence in language tasks. That is, the tokens are first independently transformed into trainable feature vectors and then processed using a transformer structure in the Symbol Encoder. Similar to the Data Encoder, the transformer structure allows for information exchange across elements in the sequence and for the construction and interpretation of mathematical structures from the symbolic guess inputs.

The processed data and symbol sequences are then concatenated and fed into the Feature Fusion block, where modality interaction and fusion occur. The data features obtain information from the symbolic input (e.g., aspects of the equation underlying the data), and the symbolic features are refined using the data (e.g., rough parameter ranges based on the data provided). After the information exchange, the fused features are ready to be decoded into corresponding outputs.

The Data Decoder constructs the operator by synthesizing two independent input sources. One source is the fused features from the Feature Fusion block, which can be interpreted as basis functions for the output space. The other source is the query time points, which are separate from the main information flow of the network. Together, the Data Decoder learns to evaluate the output basis functions at locations specified by the query time points and combines them to generate the output predictions.

The Symbol Decoder is a standard encoder-decoder transformer for sentence generation, where the fused features act as the context guiding the output expression generation. The Symbol Decoder autoregressively generates the output sentence [13, 62] from scratch, until it encounters the end-of-sentence signal. As the output sentence is starting from scratch, the model has the ability for self-correction and refinement. More specifically, it can simultaneously remove incorrect terms,

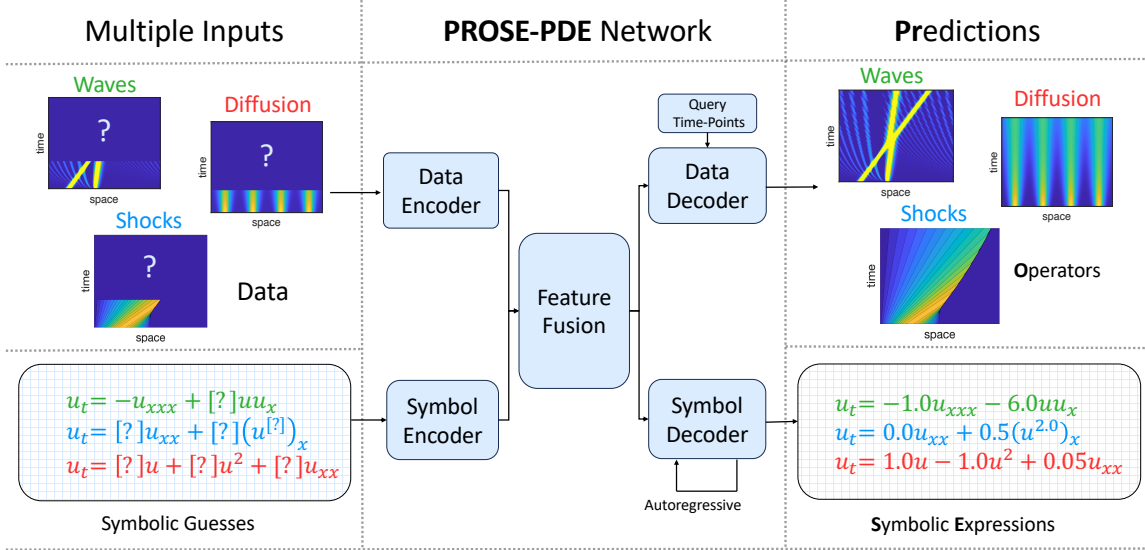


Figure 3: **PROSE-PDE Network and the Workflow.** Data input and symbolic guess input are transformed into feature vectors, which are then processed by data and symbol encoders. The processed feature vectors are combined through the feature fusion block to allow information exchange and interaction. The resulting fused features contain information from both sources and are inputs to the output structures. The upper right data decoder structure constructs the output operator based on fused features, where a separate set of query time points serve as evaluation points. PROSE-PDE generates symbolic expressions in the lower-right portion autoregressively.

generate missing terms, and identify unknown coefficients, without the need for post-processing or task-specific modules.

## 4 Experimental Setup

We study a diverse set of 20 PDEs with distinct physical features. To expand our study, we create a family for each PDE by randomly sampling PDEs' parameters from a uniform distribution within the range of  $[0.9q_c, 1.1q_c]$ , where  $q_c$  is the point-of-interest with an additional  $\pm 10\%$  value variation. This process generates a dataset containing 10.24K PDEs. To provide a closure of the PDEs, we equip each PDE with 50 initial conditions. Consequently, we create and investigate a dataset with  $10.24K \times 50 = 512K$  systems in total. We refer to Appendix A for details.

The output of the model is a prediction of the solution at (future) time points given some short-time observations and, simultaneously, a refinement of the equation guess. The input function is sampled at 16 timestamps from  $[0, t_f/2]$  and on a spatial grid of 128 points in  $[0, x_f]$ . The final time  $t_f$  may vary between different PDE, but the spatial grid length is set to  $x_f = 2$  for all systems except for the Fokker-Planck equation. We re-scale and normalize all the equations via change of variables to share the same  $t_f$  and  $x_f$  for the same experiment. The target operator maps the inputs to the solution at timestamps in the later half of the interval  $t \in [t_f/2, t_f]$  with the same spatial grid.

## 4.1 Evaluation Metrics

We use four evaluation metrics to assess the behavior of PROSE-PDE in data prediction and symbolic learning. For measuring the error for the data, the relative  $L^2$  error and the  $R^2$  score ( $R^2 = 1 - \frac{\sum_i \|\mathbf{v}_i - \mathbf{u}_i\|_2^2}{\sum_i \|\mathbf{v}_i - \text{mean}(\mathbf{v}_i)\|_2^2}$ ) are used, where  $\mathbf{u}_i$  is the  $i$ th predicted solution and  $\mathbf{v}_i$  is the  $i$ th target solution.

For the symbolic expression output, we first decode the symbolic Polish notation into trees representing functions, then the percentage of decoded output that can be transferred to valid mathematical expressions is reported. The valid expressions (which are differential operators) are applied to a set of functions randomly generated from the span of some basis set and then evaluated on a uniform space-time grid. Common choices of basis functions are polynomials and trigonometric functions [49, 50, 51, 61]. We use tensorized polynomials with degree four in space and degree two in time. More precisely, suppose  $f(\cdot)$  and  $\hat{f}(\cdot)$  are the target and PROSE-PDE generated solutions, the relative  $L^2$  error,  $\frac{\|f(P) - \hat{f}(P)\|_2}{\|\hat{f}(P)\|_2}$  is reported, where  $P$  is randomly generated. In particular, we define  $P(x, t) := P_1(x)P_2(t)$ , where  $P_1(\cdot)$  is a degree four polynomial,  $P_2(\cdot)$  is quadratic, and all the coefficients are randomly sampled from the uniform distributions on  $[-5, 5]$ . The relative  $L^2$  error is approximated on a uniform  $128 \times 64$  grid in the region  $x \in [0, 2]$  and  $t \in [0, 2]$ .

## 4.2 PROSE-PDE Modality Configurations

We explore three different modality configurations for PROSE-PDE across various tasks. The first configuration, known as the 2-to-2 model, incorporates all five structures: Data Encoder, Symbol Encoder, Feature Fusion, Data Decoder, and Symbol Decoder, as illustrated in Figure 3, and thus learns the solution operator and predicts the equations. This model demonstrates PROSE-PDE’s capability to accurately reconstruct symbolic expressions.

The second configuration, the 2-to-1 model, omits the Symbol Decoder and utilizes the remaining four components of PROSE-PDE. The 2-to-1 model is used to investigate PROSE-PDE’s ability to generalize to unseen operators with complete equations encoded as trainable symbols.

The final configuration, referred to as the 1-to-1 model, only uses the Data Encoder and Data Decoder. This model serves as a reference in assessing the significance of the symbolic component in enhancing our understanding of the operator and used in the ablation tests. For the sake of clarity, we also refer to operator I/O as data I/O and symbolic expression I/O as symbol I/O.

## 5 Results

We first evaluate the performance of PROSE-PDE by assessing two 2-to-2 model settings: the “Known” and the “Skeleton” cases. For the “Known” case, we provide the network with complete knowledge of the input equation. Thus, we expect that the network uses the encoded equation (symbol part) as an identifier and the primary evaluation is on predicting the output data. In the “Skeleton” case, the symbolic input to the network is the equation with the coefficients replaced by a placeholder, i.e. the numerical values in the equation are unknown. The objective is to simultaneously construct the data and learn the equation. We observe in Table 1 that in both cases, a low relative (data) prediction error ( $< 1.06\%$ ) and high  $R^2$  score is achieved. The “Skeleton” case

recovers the unknown equation with a low error (0.768%). Detailed results for each operator are presented in Appendix E.

Expression	Data-Noise	Unknown Coefficients	Relative (Data) Prediction Errors %	$R^2$ Scores	Relative Symbol Expression Errors %	Valid Fraction
Known	✓	✗	0.92	0.998	0.01	100.00%
Skeleton	✓	✓	1.06	0.998	0.768	99.94%

Table 1: **Experiment Settings and Results.** Data-noise: 2% additive noise on data. For the “Known” case, the equation input is known thus the relative symbol expression error measures the ability to correctly encode and re-generate the symbol space. For the “Skeleton” case, the symbolic inputs have unknown numerical values in the equation.

## 5.1 Extrapolation Studies

Classical numerical methods have reliable and highly-accurate performance on many SC tasks, e.g. solving the initial boundary value problem. Machine learning methods on the other hand have superior performance in improving repeated computations and for utilizing large amounts of data. However, the reliance on training data leaves ML’s scientific extrapolating abilities open. The concept of extrapolation has various interpretations across scientific disciplines. In computer vision, for image classification tasks it takes the form of classifying an image belonging to a class the model has not seen before. To achieve this, a model is trained on a dataset containing  $N$  classes, where each class has  $K$  samples. If we then present the model with a new dataset containing  $M$  completely different classes, and the model successfully classifies an image from this new dataset into one of those classes, this is considered a zero-shot extrapolation task.

We formulate several extrapolation settings related to SC tasks and examine the ability of PROSE-PDE (2-to-1) model to extrapolate in these settings. We first demonstrate the robustness of our model through four different types of extrapolation presented in Table 2. We will focus on the data prediction capabilities so the network is given full equation information. The following are the four settings used in **Study 1**.

**Temporal Grid.** During the testing phase, the trained model should be able to predict the output function at different points  $t$  in the domain, that is  $t \neq t_k$  where  $t_k$  is from the training set. This is the standard setting used in validating all our numerical experiments (including Table 1). This is also a measure of extrapolation since  $t$  is sampled outside of the training domain.

**Time Marching.** During the training stage, the model takes data input from  $t_{\text{in}} \in [0, 1]$  and predicts the solution in  $t_{\text{pred}} \in [1, 2]$ . During testing, we rollout the model to obtain solutions for longer intervals as follows. After obtaining predictions for times  $[1, 2]$ , we use them as the input and repeat the predictions for  $t_{\text{pred}} \in [2, t_{\text{end}}]$ . We present the results for  $t_{\text{end}} = 2.25$  in Table 2. As expected, the  $L^2$  error increases as we increase  $t_{\text{end}}$ : when  $t_{\text{end}} = 2.5$ ,  $E_{\text{pred}} = 7.09\%$ , and when  $t_{\text{end}} = 3$ ,  $E_{\text{pred}} = 10.09\%$ , where  $E_{\text{pred}}$  stands for the relative  $L^2$  prediction error.

**Out-of-Distribution (OoD).** We study the model’s ability to generalize beyond the training distribution. During testing, we sample  $q \sim \mathcal{D}'$ , where  $\mathcal{D}'$  represents a distribution larger than the



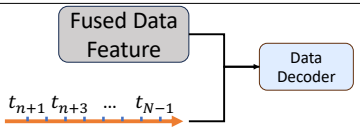
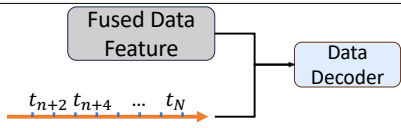
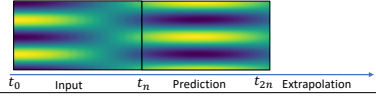
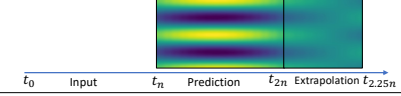

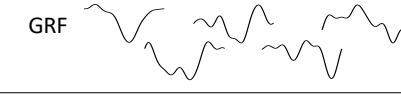
Extrapolation Settings	Training Dataset	Testing Dataset	Testing Metrics	
			Rel- $L^2$	$R^2$
Temporal Grid			0.92%	0.998
Time Marching			5.22%	0.893
Out-of-Distribution	$q_i \in (0.9 q_{i,c}, 1.1 q_{i,c})$	$q_i \in (0.8 q_{i,c}, 0.9 q_{i,c}) \cup (1.1 q_{i,c}, 1.2 q_{i,c})$	2.36%	0.986
Input Function Class	 Sinusoidal	GRF 	2.78%	0.987
Unseen Operators	$u_t + q_1(u^3)_x = q_2 u_{xx}$ $u_t + q_1 \sin(u)_x = q_2 u_{xx}$ $u_t + q_1(u^3)_x = 0$ $u_t + q_1 \sin(u)_x = 0$ $u_t + q_1(u^2)_x = 0$	Unseen during training: $u_t + q_1(u^2)_x = q_2 u_{xx}$	5.41%	0.945

Table 2: **Study 1: Various Extrapolation Results.** *Temporal Grid* (Basic setting for all experiments): Different query points (independent variables of the output functions) in training and testing. *Time marching*: Predict further steps from training. *Out-of-Distribution*: Disjoint range of free coefficients in testing. *Input Function Class*: Periodic initial condition in training, 1D gaussian random field (GRF) in testing. *Unseen Operators*: Test on an unseen equation.

training parameter distribution  $\mathcal{D}$ . Specifically, we choose the random coefficients by sampling  $q \sim \text{Uniform}(\lambda_1 q_c, \lambda_2 q_c)$ , where  $q_c$  are the points of interest. The values for  $(\lambda_1, \lambda_2)$  are chosen to be  $(0.9, 1.1)$  in the training phase, where as  $(\lambda_1, \lambda_2) = (0.8, 0.9), (1.1, 1.2)$  are used during testing. This shows the model can be used to predict solutions even if the coefficients are not seen in training.

**Input Function Class.** We extend the concept of OoD by changing the generating function, i.e., testing samples are produced using a distinct generation function  $\mathcal{G}'$  along with a different parameter distribution  $\mathcal{D}'$ . For Table 2, in training we used periodic initial conditions generated as sums of sinusoidal functions, whereas in testing, the initial conditions are generated using Gaussian random fields (GRF). The testing case has larger variations and is thus less regular than the training set. We refer to Appendix A and Table 6 for details.

**Unseen Operators.** We aim to assess whether the trained neural operator can adapt to operators unseen during training. Specifically, we test the model with operators  $G'$  that are not included in the training operator set  $\{G_i\}_{i=1}^{N_o}$ . We use five operator families  $\{G_i\}_{i=1}^5$  in training, and evaluate the operator on the viscous Burgers' equation  $G'$  in testing. We randomly sample the free parameters of the PDEs and generate 128K systems for training and 102.4K systems for testing. These results show the capability of PROSE-PDE to learn a new operator  $G'$  without any fine-tuning.

## Extrapolation of Physical Features Studies

In this section, we demonstrate the EPF property of the proposed network. We show that PROSE-PDE can transfer unknown physics features by learning similarities from other operators and predict



such physical features even in the absence of exposure during the training phase.

In the Input Function Class extrapolation tests, changes in the initial conditions can lead to new phenomena in the outputs. However, this directly depends on the equation and the input function class. As an example, consider the case where the training set is generated by randomizing the (finite) Fourier series and where we test on the Riemann problem, e.g., the initial data in the testing phase is generated by a step function:

$$f(x) = \begin{cases} 0 & \text{if } x \in [-\pi, 0] \\ 1 & \text{if } x \in [0, \pi]. \end{cases}$$

Then although this initial condition was not seen during training, it could be approximated by, for example,  $h(x) = \frac{1}{2} + \frac{2}{\pi} \sin(x) + \frac{2}{3\pi} \sin(3x)$  which may be seen in the training phase. While still an extrapolation test, the testing samples may bear some resemblance to the training samples. In Study 2 and 3, we propose a more demanding extrapolation test. Specifically, we pick the testing generator such that the output functions exhibit fundamentally different physical characteristics.

**Study 2: Transferring Physical Features:** The two prevalent physical phenomena that are observed in conservation laws are shocks and rarefaction waves. The objective of Study 2 is to investigate PROSE-PDE’s ability to generalize these physical phenomena between distinct equations. To test this, we design a series of experiments in which we change the proportion of shocks and rarefaction waves sampled in training data and measure the model’s ability to predict unseen rarefaction waves in testing.

The data is generated using the Riemann problem with initial conditions located within the interval  $[0, 1]$ , and with homogeneous Neumann boundary conditions. All testing is done on the rarefaction setting for the viscous Burgers’ equation, which is not used in training. The shock setting for the viscous Burgers’ equation is included in the training set to help link it to other conservation laws, i.e., help with the transferring process. For instance, in Experiment 1 (first row of Table 3) the training dataset includes rarefaction waves from all equations except the target equation and shocks only from the target equation. Conversely, in Experiment 5 (last row of Table 3), the only equation in the training dataset exhibiting a rarefaction solution is the Cosine Flux equation. Note that each row of Table 3 includes about 3K randomly generated systems in the listed types for a total of 153.6K systems for training and 20.48K random systems for testing.

We observe from the Table 3 that PROSE-PDE can construct rarefaction waves for viscous Burgers’ systems without directly observing them. This shows the model’s generalization and EPF capabilities. Specifically, PROSE-PDE likely learns the mechanism behind the rarefaction wave based on the training data of other systems and generalizes it to the viscous Burgers’ setting.

**Remark 5.1.** *To check if the network memorizes rarefaction features from other equations and applies those directly to the target equation during testing, we measure the similarity of the training systems as follows. We generate the solutions for the target equation and the solutions of the other randomly sampled systems by reusing the same initial conditions. We define the similarity  $e_i$  by*

$$e_i = \|G_{target}[u](\cdot) - G_i[u](\cdot)\|_2 / \|G_{target}[u](\cdot)\|_2, \tag{5.1}$$

where  $u$  is the initial condition (leading to rarefaction waves),  $G_{target}$  is the true viscous Burgers’ solution operator, and  $G_i$  is the true solution operator of the  $i$ th training equations.









Exp No.	Training Sample						Testing Sample	Rel- $L^2$ Error
	Viscous Burgers'	Burgers'	Viscous Cubic Flux	Cubic Flux	Viscous Cosine Flux	Cosine Flux	Viscous Burgers'	
Exp 1								0.49%
Exp 2				Rarefaction				0.54%
Exp 3			Shock					
Exp 4								
Exp 5								2.34%

Table 3: **Study 2: Transferring Physical Features.** Each row in the table represents a distinct experiment. The purple region indicates that the training data corresponding to the listed PDE type (the columns) consists of shock solutions, while the blue region indicates that the training data corresponding to the listed PDE type consists of rarefaction waves. As a reference, in Exp. 5 the prediction error 2.34% is lower than directly using a fitted cosine flux as a prediction, which yields 3.59% error.

The computed similarities over the training equations are 1.38% for Burgers', 14.16% for viscous cubic flux, 13.66% for cubic flux, 1.85% for viscous cosine flux, and 3.59% for cosine flux, respectively. Note that the prediction errors for each experiment in Table 3 are consistently lower than the minimum of these values over the corresponding experiment. This suggests that the PROSE-PDE model extrapolates the data rather than merely replicating the best (training) operator.

**Study 3: Generalizing to Multiple Shocks:** We examine the model's ability to generalize single shocks to multiple shock interactions, specifically concentrating on four PDE types all in polynomial flux forms. In particular, PROSE-PDE is trained using data that leads to one shock (purple region of Table 4) or multiple shocks (blue region of Table 4). The initial conditions are set within the range  $[0, 1]$ , with homogeneous Neumann boundary conditions.

Table 4 details each experiment setting and demonstrates the model's ability to accurately resolve the behavior of two shocks in the viscous cubic flux setting, even when the exact physical feature is not present during training. We verify that the network is not memorizing the training data by checking the similarities as defined in Equation (5.1), with  $G_{\text{target}}$  representing the true viscous cubic flux solution operator. The recorded similarities are 1.15% for cubic flux, 16.06% for viscous Burgers', and 15.74% for Burgers' equation. The prediction errors shown in Table 4 remain consistently below these similarity values, indicating that PROSE-PDE effectively extrapolates to two shock interactions for the target operator, rather than simply repeating what is observed in the training dataset.

## 5.2 Ablation Studies

The PROSE-PDE model is a bi-modal to bi-modal model, specifically, it maps the data and symbols to predicted data and symbols. In this section, we will show that both symbolic input and output







Exp No.	Training Sample				Testing Sample	Rel- $L^2$ Error
	Viscous Cubic Flux	Cubic Flux	Viscous Burgers'	Burgers'	Viscous Cubic Flux	
Exp 1						0.77%
Exp 2						1.02%
Exp 3						2.35%
Exp 4						4.48%

Table 4: **Study 3: Shock Interactions.** Each row in the table represents a distinct experiment. For each listed PDE type (the columns), a purple region indicates that the training data of this PDE type are single-shock solutions, while a blue region indicates that the training data of this PDE type are multi-shock solutions. As a reference, in Exp. 3 the prediction error 2.35% is lower than directly using the Burgers’ equation with the same initial conditions, which yields 16.06% error.

modalities enhance the learning of data prediction.

### 5.2.1 Ablation Study: Symbolic Encoder-Decoder

In PROSE-PDE, trainable tokens are used to represent the input equation and thus help to identify the operator. In this section, our objective is to illustrate that such representation can effectively discern between different operators. We conduct a comparison between the PROSE-PDE 2-to-1 models, the model utilizing only the data modality (1-to-1 model), and two single operator learning models (FNO and DeepONet). Note that the input functions are only sampled at the initial timestamp, thus the learning problem is ill-posed for a model relying solely on the data modality. Since the 2-to-1 model leverages the equation information, we expect the learning to remain well-posed. As shown in Table 5, PROSE-PDE 2-to-1 (first row) produces reliable predictions with small errors (1.03%) even when supplied with only the initial conditions as data input, showing that benefit of the symbol encoder.

To test the model’s dependency on the number of timestamps for the input function, we test the PROSE-PDE 2-to-2 model with “Skeleton” inputs and vary the number of timestamps, i.e., the input size. Figure 4 shows the consistent behavior of our model over the input size, which indicates the stability induced by the Symbol Encoder-Decoder in providing additional information for the data prediction.

### 5.2.2 Ablation Study: Loss Weights

Lastly, we want to demonstrate the robustness of the model with respect to the importance of minimizing the loss for each output modality. This is done by varying the weights between the two output losses. Specifically, in Figure 5, we change the weights assigned to the data and symbol losses.

Modality	Testing Metrics for Data Prediction	
	Rel- $L^2$ error (%)	$R^2$ score
Data/Symbol encoder + Fusion + Data decoder (2-to-1)	1.03/3.03/3.52	0.997/0.982/0.972
Data encoder + Data decoder (1-to-1)	29.5/41.21/30.86	-1.095/-0.56/-0.99
FNO	34.86/37.32/35.42	-2.82/-1.82/-2.28
DeepONet	32.07/37.88/32.82	-2.08/-1.22/-2.00

Table 5: **Ablation Study:** Comparing the 2-to-1 model to data-only architecture and two single operator learning networks (FNO and DeepONet) using the data-prediction error. Note that the input functions are only sampled at initial timestep to predict all future states. The error are indicated as Extrapolation in Temporal Grid/ Input function class/ Out-of-Distribution as described in Section 5.1. The results indicate the need for symbolic information to discern operators in MOL.

We observe a slight increase in the error for the data output when we reduce the weight ratio (data weight over symbol weight) from 5 to 0.2. However, the overall data-prediction error remains low. This suggests that the output symbol modality contributes to the overall improvement through the Fusion layers. That is, by including some symbolic information, the model’s data prediction remains resilient to changes in the training hyperparameters.

To illustrate this, Figure 6 shows the gradient information flow for the PROSE-PDE model during the training phase. Assigning a larger weight to the symbol component enhances the learning of the lower portion of the PROSE-PDE model. However, this adjustment negatively impacts the Data Decoder for data learning. We hypothesize that the inclusion of the symbol modality enhances the learning of the Fusion structure, thereby generating a stronger encoding for the operator (data prediction). Consequently, we only observe a slight decrease in performance for data prediction which underscores the advantages of the symbol modality.

## 6 Discussion

The PROSE-PDE approach is a bi-modality to bi-modality model for solving forward and inverse tasks in multi-operator learning of spatiotemporal systems. The main focus of this work is on presenting and testing an architecture for a PDE foundation model for time-dependent nonlinear partial differential equations that can generalize. Through detailed experiments, the PROSE-PDE approach is shown to generalize in various settings without the need of fine-tuning. Most importantly, the model is able to extrapolate some physical features as verified by testing the extrapolation capa-

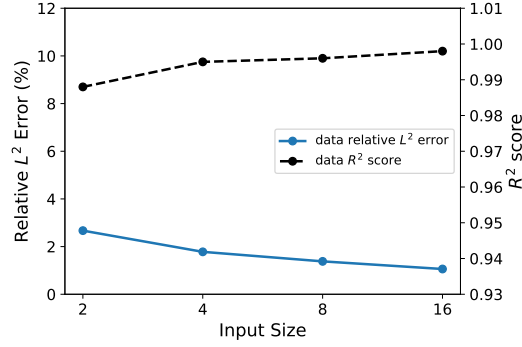


Figure 4: Comparing the PROSE-PDE 2-to-2 model with varying amounts of input timestamps, i.e. input sizes and fixed output grid, i.e.  $t > t_f/2$ . The equations are in “Skeleton” form and thus have unknown coefficients.

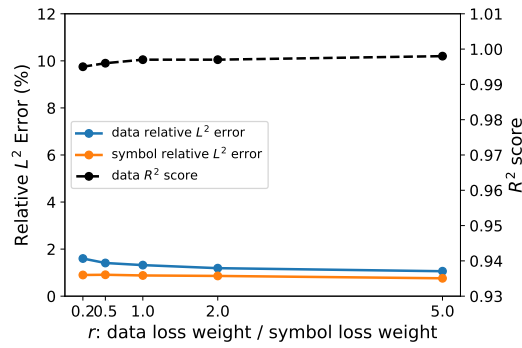


Figure 5: Relative  $L^2$  errors and  $R^2$  scores with varying ratio of data/symbol loss weight. The results are reported using the model with the lowest symbol error in the validation set.

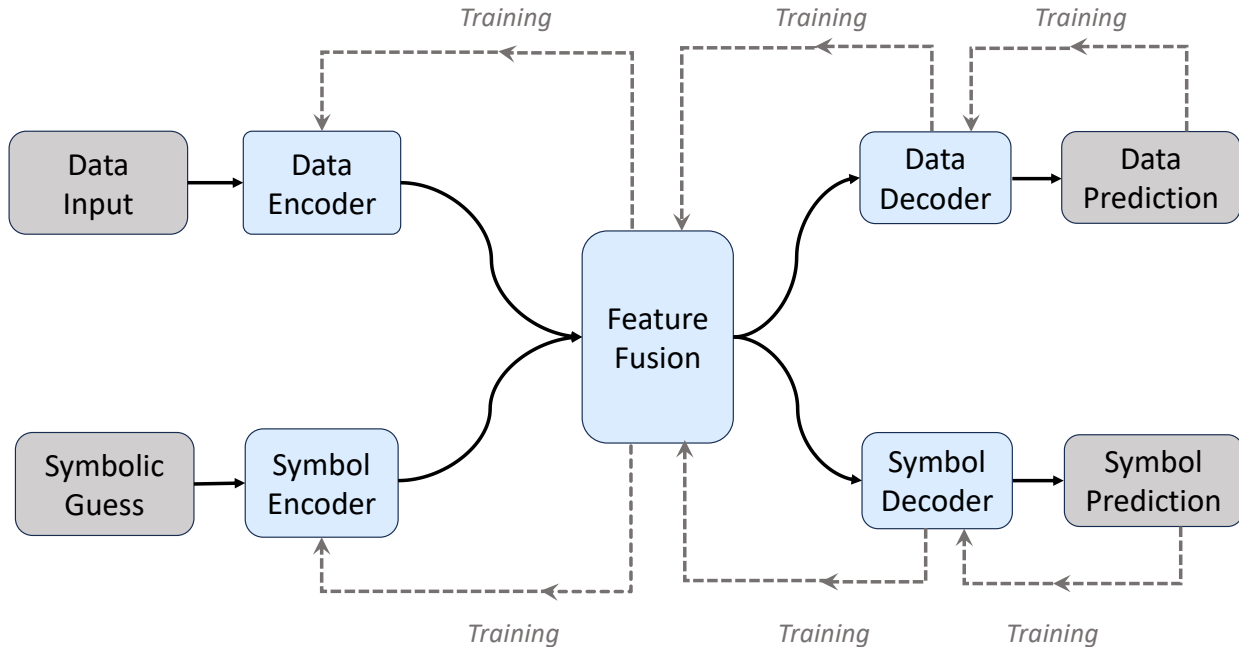


Figure 6: **Gradient Information Flow:** We illustrate the gradient information flow for the PROSE-PDE model during the training phase. The training phase couples the modalities’ parameters.

bilities for rarefactions and multi-shock interaction. We expect stronger capabilities with a larger training set. The ablation results demonstrated the importance of multi-modal information for generating consistent and robust results. A future direction is to extend the PROSE-PDE architecture to multi-dimensional nonlinear partial differential equations and to include non-time dependent PDE.

## Data Availability Statement

The datasets generated and analyzed during the current study, including materials characterization data and simulation results, are available at <https://github.com/felix-lyx/prose>.

## Acknowledgement

J. Sun, Y. Liu, and H. Schaeffer were supported in part by an AFOSR MURI FA9550-21-1-0084 and NSF DMS-2331033. Z. Zhang was supported in part by NSF DMS-2331033.

## References

- [1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [2] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

- [3] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.
- [4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [5] Francois Charton. Linear algebra with transformers. *arXiv preprint arXiv:2112.01898*, 2022.
- [6] 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.
- [7] 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.
- [8] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.
- [9] Stephane d’Ascoli, Pierre-Alexandre Kamienny, Guillaume Lample, and Francois Charton. Deep symbolic regression for recurrent sequences. *arXiv preprint arXiv:2201.04600*, 2022.
- [10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [11] Patricio Clark Di Leoni, Lu Lu, Charles Meneveau, George Em Karniadakis, and Tamer A Zaki. Neural operator prediction of linear instability waves in high-speed boundary layers. *Journal of Computational Physics*, 474:111793, 2023.
- [12] Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209. Association for Computational Linguistics, 2016.
- [13] Luciano Floridi and Massimo Chiriatti. Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30:681–694, 2020.
- [14] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.
- [15] Viktor Holubec, Klaus Kroy, and Stefano Steffenoni. Physically consistent numerical solver for time-dependent fokker-planck equations. *Phys. Rev. E*, 99:032117, Mar 2019.
- [16] Amanda A Howard, Mauro Perego, George E Karniadakis, and Panos Stinis. Multifidelity deep operator networks. *arXiv preprint arXiv:2204.09157*, 2022.
- [17] Shima Imani, Liang Du, and Harsh Shrivastava. Mathprompter: Mathematical reasoning using large language models. *arXiv preprint arXiv:2303.05398*, 2023.



- [18] Zhongyi Jiang, Chunmei Wang, and Haizhao Yang. Finite expression methods for discovering physical laws from data. *arXiv preprint arXiv:2305.08342*, 2023.
- [19] Zhongyi Jiang, Min Zhu, Dongzhuo Li, Qiuzi Li, Yanhua O Yuan, and Lu Lu. Fourier-mionet: Fourier-enhanced multiple-input neural operators for multiphase modeling of geological carbon sequestration. *arXiv preprint arXiv:2303.04778*, 2023.
- [20] Pengzhan Jin, Shuai Meng, and Lu Lu. Mionet: Learning multiple-input operators via tensor product. *SIAM Journal on Scientific Computing*, 44(6):A3490–A3514, 2022.
- [21] Pierre-Alexandre Kamienny, Stephane d’Ascoli, Guillaume Lample, and Francois Charton. End-to-end symbolic regression with transformers. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.
- [22] Wonjae Kim, Bokyung Son, and Ildoo Kim. Vilt: Vision-and-language transformer without convolution or region supervision. In *International Conference on Machine Learning*, pages 5583–5594. PMLR, 2021.
- [23] Wing Tat Leung, Guang Lin, and Zecheng Zhang. Nh-pinn: Neural homogenization-based physics-informed neural network for multiscale problems. *Journal of Computational Physics*, page 111539, 2022.
- [24] Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. Visualbert: A simple and performant baseline for vision and language. *arXiv preprint arXiv:1908.03557*, 2019.
- [25] Ruilong Li, Shan Yang, David A Ross, and Angjoo Kanazawa. Ai choreographer: Music conditioned 3d dance generation with aist++. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13401–13412, 2021.
- [26] 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.
- [27] Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries. *Journal of Machine Learning Research*, 24(388):1–26, 2023.
- [28] 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.
- [29] 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.
- [30] Senwei Liang and Haizhao Yang. Finite expression method for solving high-dimensional partial differential equations. *arXiv preprint arXiv:2206.10121*, 2022.
- [31] 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.

- [32] Guang Lin, Christian Moya, and Zecheng Zhang. Learning the dynamical response of non-linear non-autonomous dynamical systems with deep operator neural networks. *Engineering Applications of Artificial Intelligence*, 125:106689, 2023.
- [33] Yuxuan Liu, Zecheng Zhang, and Hayden Schaeffer. PROSE: Predicting operators and symbolic expressions using multimodal transformers. *arXiv preprint arXiv:2309.16816*, 2023.
- [34] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. *Advances in neural information processing systems*, 32, 2019.
- [35] 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.
- [36] 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.
- [37] Lu Lu, Raphaël Pestourie, Steven G Johnson, and Giuseppe Romano. Multifidelity deep neural operators for efficient learning of partial differential equations with application to fast inverse design of nanoscale heat transport. *Physical Review Research*, 4(2):023210, 2022.
- [38] Shunyuan Mao, Ruobing Dong, Lu Lu, Kwang Moo Yi, Sifan Wang, and Paris Perdikaris. Ppdonet: Deep operator networks for fast prediction of steady-state solutions in disk–planet systems. *The Astrophysical Journal Letters*, 950(2):L12, 2023.
- [39] Zhiping Mao, Lu Lu, Olaf Marxen, Tamer A Zaki, and George Em Karniadakis. Deepm&mnet for hypersonics: Predicting the coupled flow and finite-rate chemistry behind a normal shock using neural-network approximation of operators. *Journal of computational physics*, 447:110698, 2021.
- [40] Michael McCabe, Bruno Régaldo-Saint Blancard, Liam Holden Parker, Ruben Ohana, Miles Cranmer, Alberto Bietti, Michael Eickenberg, Siavash Golkar, Geraud Krawezik, Francois Lanusse, et al. Multiple physics pretraining for physical surrogate models. *arXiv preprint arXiv:2310.02994*, 2023.
- [41] Christian Moya, Amirhossein Mollaali, Zecheng Zhang, Lu Lu, and Guang Lin. Conformalized-deepoNet: A distribution-free framework for uncertainty quantification in deep operator networks. *arXiv preprint arXiv:2402.15406*, 2024.
- [42] Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, et al. FourCastNet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.
- [43] H. A. Pogorzelski. Review: Jan lukasiewicz, Jerzy slupecki, panstwowe wydawnictwo, remarks on nicod’s axiom and on ”generalizing deduction”. *Journal of Symbolic Logic*, 30(3):376–377, 1965.

- [44] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.
- [45] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [46] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3, 2022.
- [47] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International conference on machine learning*, pages 8821–8831. Pmlr, 2021.
- [48] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.
- [49] 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.
- [50] Hayden Schaeffer and Scott G McCalla. Sparse model selection via integral terms. *Physical Review E*, 96(2):023302, 2017.
- [51] 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.
- [52] Junhong Shen, Tanya Marwah, and Ameet Talwalkar. Ups: Towards foundation models for pde solving via cross-modal adaptation. *arXiv preprint arXiv:2403.07187*, 2024.
- [53] Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. Videobert: A joint model for video and language representation learning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 7464–7473, 2019.
- [54] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566. Association for Computational Linguistics, 2015.
- [55] Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Daniel MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. Pdebench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems*, 35:1596–1611, 2022.
- [56] Hao Tan and Mohit Bansal. Lxmert: Learning cross-modality encoder representations from transformers. *arXiv preprint arXiv:1908.07490*, 2019.
- [57] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

- [58] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [59] Yao-Hung Hubert Tsai, Shaojie Bai, Paul Pu Liang, J Zico Kolter, Louis-Philippe Morency, and Ruslan Salakhutdinov. Multimodal transformer for unaligned multimodal language sequences. In *Proceedings of the conference. Association for Computational Linguistics. Meeting*, volume 2019, page 6558. NIH Public Access, 2019.
- [60] Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. Language models don’t always say what they think: unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36, 2024.
- [61] Silviu-Marian Udrescu and Max Tegmark. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.
- [62] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [63] 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.
- [64] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google’s neural machine translation system: Bridging the gap between human and machine translation, 2016.
- [65] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015.
- [66] Peng Xu, Xiatian Zhu, and David A Clifton. Multimodal learning with transformers: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.
- [67] 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.
- [68] 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.
- [69] 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.

- [70] Zhanhong Ye, Xiang Huang, Leheng Chen, Hongsheng Liu, Zidong Wang, and Bin Dong. Pdeformer: Towards a foundation model for one-dimensional partial differential equations. *arXiv preprint arXiv:2402.12652*, 2024.
- [71] Minglang Yin, Nicolas Charon, Ryan Brody, Lu Lu, Natalia Trayanova, and Mauro Maggioni. Dimon: Learning solution operators of partial differential equations on a diffeomorphic family of domains. *arXiv preprint arXiv:2402.07250*, 2024.
- [72] N. J. Zabusky and M. D. Kruskal. Interaction of "solitons" in a collisionless plasma and the recurrence of initial states. *Phys. Rev. Lett.*, 15:240–243, Aug 1965.
- [73] Zecheng Zhang. Modno: Multi operator learning with distributed neural operators. *arXiv preprint arXiv:2404.02892*, 2024.
- [74] Zecheng Zhang, Wing Tat Leung, and Hayden Schaeffer. Belnet: basis enhanced learning, a mesh-free neural operator. *Proceedings of the Royal Society A*, 479(2276):20230043, 2023.
- [75] Zecheng Zhang, Wing Tat Leung, and Hayden Schaeffer. A discretization-invariant extension and analysis of some deep operator networks. *arXiv preprint arXiv:2307.09738*, 2023.
- [76] Zecheng Zhang, Christian Moya, Wing Tat Leung, Guang Lin, and Hayden Schaeffer. Bayesian deep operator learning for homogenized to fine-scale maps for multiscale pde. 2023.
- [77] Zecheng Zhang, Christian Moya, Lu Lu, Guang Lin, and Hayden Schaeffer. D2no: Efficient handling of heterogeneous input function spaces with distributed deep neural operators. *arXiv preprint arXiv:2310.18888*, 2023.
- [78] Min Zhu, Shihang Feng, Youzuo Lin, and Lu Lu. Fourier-deeponet: Fourier-enhanced deep operator networks for full waveform inversion with improved accuracy, generalizability, and robustness. *arXiv preprint arXiv:2305.17289*, 2023.
- [79] Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems*, 36, 2024.

## A Dataset Details

### A.1 PDE Types

The parameter-of-interest is listed for each type. Note that the training and testing data randomizes the values of the parameters.

#### A.1.1 Diffusion Equation

$$u_t - cu_{xx} = 0$$

where  $c = 3 \times 10^{-3}$ ,  $t_{final} = 2$ .

### A.1.2 Porous Medium Equation

$$u_t - (u^m)_{xx} = 0$$

where  $m = 2, 3, 4$  and  $t_{final} = 0.1$ .

### A.1.3 Klein-Gordon Equation

$$u_{tt} - c^2 u_{xx} + m^2 c^4 u = 0$$

where  $c = 1$ ,  $m = 0.1$ , and  $t_{final} = 1$ .

### A.1.4 Sine-Gordon Equation

$$u_{tt} - u_{xx} + c \sin(u) = 0$$

where  $c = 1$ , and  $t_{final} = 1$ .

### A.1.5 Cahn-Hilliard Equation

$$u_t + \epsilon^2 u_{xxxx} + 6(uu_x)_x = 0$$

where  $\epsilon = 0.01$ , and  $t_{final} = 0.5$ .

### A.1.6 Korteweg–De Vries (Kdv) Equation

$$u_t + \delta^2 u_{xxx} + uu_x = 0$$

where  $\delta = 0.022$ , and  $t_{final} = 1$ .

### A.1.7 Advection Equation

$$u_t + \beta u_x = 0$$

where  $\beta = 0.5$  and  $t_{final} = 2$ .

### A.1.8 Wave Equation

$$u_{tt} - \beta u_{xx} = 0$$

where  $\beta = 0.5$  and  $t_{final} = 1$ .

### A.1.9 Diffusion Reaction Equation

$$u_t - \nu u_{xx} - \rho R(u) = 0$$

where  $\nu = 3 \times 10^{-3}$ , and  $\rho = 1$  for  $R = R_1, R_3, R_4$ ; and  $\rho = 0.1$  for  $R_2$ , and  $t_{final} = 2$ .

$$R_1(u) = u(1 - u)$$

$$R_2(u) = u$$

$$R_3(u) = u^2(1 - u)$$

$$R_4(u) = u^2(1 - u)^2$$

### A.1.10 Viscous Conservation Law

$$u_t + kf(u)_x - \frac{\epsilon}{\pi} u_{xx} = 0$$

where  $k = 1$ ,  $\epsilon = 0.01$ , and  $t_{final} = 2$ .

$$f_1(u) = \frac{1}{2}u^2 \quad \text{Burgers' equation}$$

$$f_2(x) = u$$

$$f_3(x) = \frac{1}{3}u^3$$

$$f_4(x) = \sin(x)$$

### A.1.11 Inviscid Conservation Law

$$u_t + kf(u)_x = 0$$

where  $k = 1$  and  $t_{final} = 2$ .

$$f_1(u) = \frac{1}{2}u^2 \quad \text{Inviscid Burgers' equation}$$

$$f_2(x) = \frac{1}{3}u^3$$

$$f_3(x) = \sin(x)$$

### A.1.12 Fokker-Planck Equation

$$u_t = Du_{xx} - \frac{D}{k_B T} (\nabla U(x)u)_x$$

where  $D = \frac{k_B T}{\gamma}$ , where  $k_B \approx 1.380649 \times 10^{-23}$  is the Boltzmann constant,  $T = 300$  is absolute temperature, and  $\gamma = 6\pi\eta r$  represent the drag coefficient,  $\eta = 10^{-3}$  is the fluid viscosity (randomized), and  $r = 0.1 \times 10^{-6}$ .  $U(x) = c \cos\left(\frac{x}{L}\right)$ , where  $c = 5 \times 10^{-21}$ , and  $L = 0.1 \times 10^{-6}$ . Set  $t_{final} = 0.1$ , and  $x_{final} = 2 \times 10^{-6}$ .



## A.2 Initial Conditions

We mainly consider periodic boundary conditions unless specified, and we use different types of initial conditions for different types of equations:

### Super-position of sinusoidal waves

This is derived from PDEBench [55]:

$$u_0(x) = \sum_{k=k_1, \dots, k_N} A_i \sin(k_i x + \phi_i) \quad (\text{A.1})$$

where  $k_i = 2\pi n_i / L_x$ ,  $n_i$  is randomly selected integers from  $[1, n_{max}]$ ,  $L_x$  is the spatial domain size. The amplitude  $A_i$  is random float uniformly chosen in  $[0, 1]$ , and  $\phi_i$  is the randomly chosen phase in  $(0, 2\pi)$ . For all equations except advection and wave equation, after calculating (A.1), we enforced absolute value with random signature and the window function with 10% probability.

### Gaussian Process

$$u_0(x) \sim \mathcal{N}(0, K_x) \quad (\text{A.2})$$

where the covariance matrix  $K_x$  is obtained by the RBF kernel with  $x_1 = x_2 = x$ , and the RBF kernel is described as

$$k_{RBF}(x_1, x_2) = \sigma^2 \exp\left(-\frac{\|x_1 - x_2\|^2}{2l^2}\right) \quad (\text{A.3})$$

### Gaussian Distribution

$$u_0(x) = \sum_{i=1}^N A_i \exp\left(-\frac{|x - \mu_i|^2}{2\sigma_i^2}\right) \quad (\text{A.4})$$

### Uniform Distribution

$$u_0(x) \sim \mathbf{Uniform}(x_l, x_r) \quad (\text{A.5})$$

### Quadratic function

$$u_0(x) = \max\left(-A \frac{(x - \mu)^2}{2\sigma^2} + A, 0\right) \quad (\text{A.6})$$

### Periodization and normalization

We enforced periodicity for initial condition  $u_0(x)$  by removing the linear function  $l(x)$  that passes through the endpoints of the domain, hence the modified initial condition is given by

$$u'_0(x) = u_0(x) - l(x). \quad (\text{A.7})$$

We also normalize the initial condition in two distinct ways:

- Adjust  $u'_0(x)$  so that the range falls within  $(0, u_{max})$ .
- When the initial condition is represented by a probability distribution, we adjusted  $u_0(x)$  such that the sum of probability is 1.

Equation type	Training Initial Condition	Testing Initial Condition
Heat	(A.1): $n_{max} = 2$	(A.2): $\sigma = 1, l = 0.2$
Diff-React		
Klein-Gordon		
Sine-Gordon		
Cahn-Hilliard		
Viscous Conservation	(A.4): $N = 2$	(A.5)
Inviscid Conservation		
Kdv		
Advection	(A.4): $N = 1$	(A.6)
Wave		
Fokker-Planck		
Porous medium		

Table 6: Choice of Training and Evaluation Initial Condition for different types of equations

Equation type	Generator
Heat	Method of Line
Klein-Gordon	
Sine-Gordon	
Porous medium	
Cahn-Hilliard	
Diff-React	PDEBench [55]
Viscous Conservation	
Inviscid Conservation	
Advection	Exact solution defined by IC
Wave	
Kdv	Fourier Spectral Method [72]
Fokker-Planck	Matrix Numerical Method [15]

Table 7: Solvers for different types of equations

### A.3 Solvers

As detailed in Table 7, we use different solvers for different types of equations. For the diffusion-reaction equation and all types of conservation laws, we employ PDEBench [55]. The Matrix Numerical Methods (MNM) introduced in [15] are used for solving the Fokker-Planck equation, while the pseudo-spectral method from [72] is applied to the KdV equation. For advection and wave equations, we utilize the exact solution defined by the initial conditions. The method of lines, which discretizes the PDE in space and solves the ODE in time, is used for the rest of the equations.

## B Architecture Details

Our network uses hierarchical attention for feature processing and fusion, and two transformer decoders for two downstream tasks. Figure 7 provides an overview of the architecture. The PROSE-PDE architecture contains five main components trained end-to-end: data encoder, symbol encoder, feature fusion, data decoder, and symbol decoder.

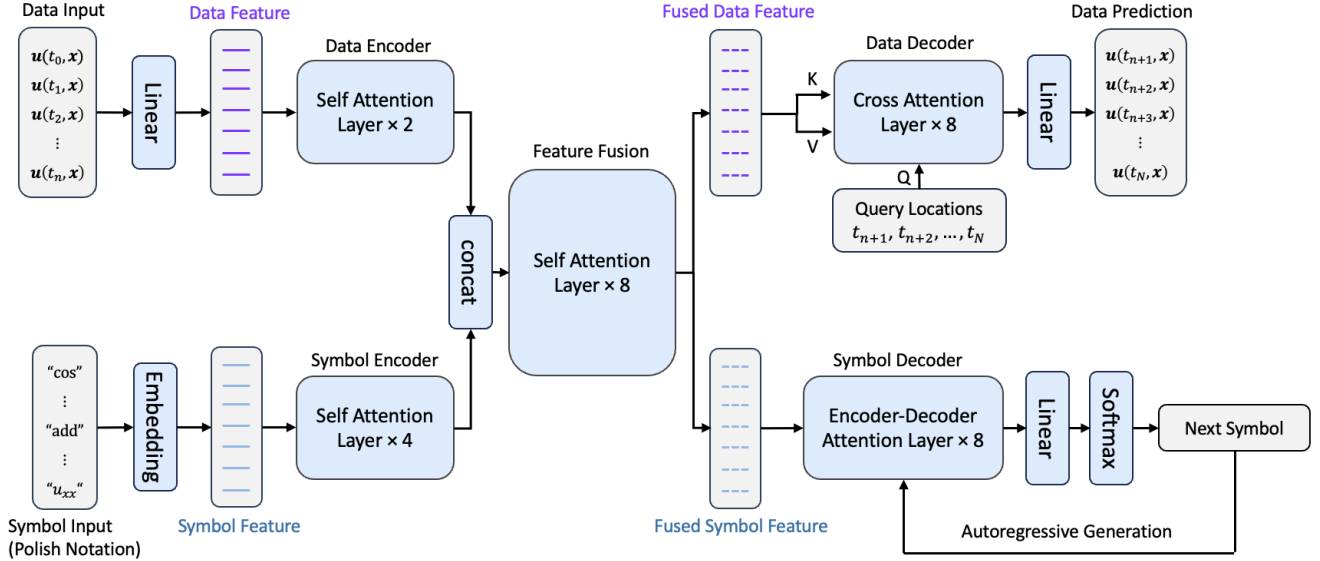


Figure 7: **PROSE-PDE Architecture Details.** Data Input and Symbol Input are embedded into Data Feature and Symbol Feature respectively before encoding and fusion through Feature Fusion. PROSE-PDE uses Cross-Attention to construct the operator (upper-right structure) from Fused Data Feature, and evaluate it at Query Locations. PROSE-PDE generates symbolic expressions in the lower-right portion autoregressively. Attention blocks are displayed in Appendix C, where each layer also includes a feedforward network.

## B.1 Encoders

Two separate transformer encoders are used to obtain domain-specific features. Given numerical data inputs and symbolic equation guesses (possibly empty or erroneous), the data encoder and symbol encoder first separately perform feature aggregation using self-attention. For a data input sequence  $\mathbf{u}(t_0), \dots, \mathbf{u}(t_n)$ , each element  $\mathbf{u}(t_i)$ , together with its time variable  $t_i$ , goes through a linear layer to form the Data Feature (purple feature sequence in Figure 7). PROSE-PDE then uses self-attention to further process the Data Feature, where the time variables  $t_i$  serve as the positional encoding.

The symbolic input (in Polish notation) is a standard word sequence, which can be directly processed with self-attention layers. The word embedding (for operations, sign, mantissa, etc.) is randomly initialized and trainable. Sinusoidal positional encoding [62] is used for the symbol encoder.

## B.2 Feature Fusion

Hierarchical attention (multi-stream to one-stream) is used in this model for feature fusion. Separately-processed data and symbol features are concatenated into a feature sequence, and further processed through self-attention layers where modality interaction occurs. Following [22], a learnable modality-type embedding is added to the fused features, signaling the source modality of each token. Positional encoding is not needed since it is already included in the individual encoders.

### B.3 Data Decoder

The data decoder constructs the operator via the cross-attention mechanism, establishing a link between the input-encoded time sequence (fused data features) and the output functions. The query locations, representing the independent variables of these output functions, serve as the evaluation points. Importantly, these query locations operate independently of each other, meaning that assessing the operator at one point,  $t_i$ , does not impact the evaluation of the operator at another point,  $t_j$ . As a result, the time and space complexity scales linearly with the number of query locations. In addition, since the evaluation points are independent of the network generation, this resembles the philosophy of the branch and trunk nets, see Operator Learning Structure in Section 3.

### B.4 Symbol Decoder

The symbol decoder is a standard encoder-decoder transformer, where the fused symbol feature is the context for generation. The output equation is produced using an autoregressive approach [13, 62]: it starts with the start-of-sentence token and proceeds iteratively, generating each term of the equation based on prior predictions, until it encounters the end-of-sentence token for that specific equation. During evaluation time, greedy search (iterative selection of symbol with maximum probability) is used for efficient symbol generation. While beam search [64] can be used to improve the performance (e.g. percentage of valid expression outputs), we empirically find that greedy search is sufficient for obtaining valid mathematical expressions using the Polish notation formulation.

## C Preliminary

### C.1 Transformers

A transformer operates on the principle of attention, adept at identifying long-range dependencies within data sources, as noted by [2, 8, 62]. It processes input by assigning varying degrees of importance to different segments of the data sequence, thereby focusing or “attending” to particular portions of the input for decision-making or output generation. The standard transformer model employs a self-attention framework, as described by [1, 65], which allows it to discern complex patterns in extensive time series data.

Specifically, let us denote the input time series data as  $X \in \mathbb{R}^{n \times d}$ , where  $n$  is the number of time steps and  $d$  is the dimension of each element in the time series. Self-attention first computes the projections: query  $Q = XW^Q$ , key  $K = XW^K$  and value  $V = XW^V$ , where  $W^Q \in \mathbb{R}^{d \times d_k}$ ,  $W^K \in \mathbb{R}^{d \times d_k}$ , and  $W^V \in \mathbb{R}^{d \times d_v}$ . It then outputs the context  $C \in \mathbb{R}^{n \times d_v}$  via

$$C = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V, \quad (\text{C.1})$$

where the softmax function is calculated over all entries of each row. Self-attention discovers relationships among various elements within a time sequence. Predictions often depend on multiple data sources, making it crucial to understand the interactions and encode various time series data (see Section 2.2 for details). The self-attention is also used in the development of the cross-attention mechanism [25, 34, 59]. Given two input time series data  $X, Y$ , cross-attention computes the query, key, and value as  $Q = XW^Q$ ,  $K = YW^K$ , and  $V = YW^V$ . In the case where  $Y$  represents the output of a decoder and  $X$  represents the output of an encoder, the cross-attention, which directs its focus from  $X$  to  $Y$ , is commonly referred to as encoder-decoder attention [62]. Encoder-decoder attention

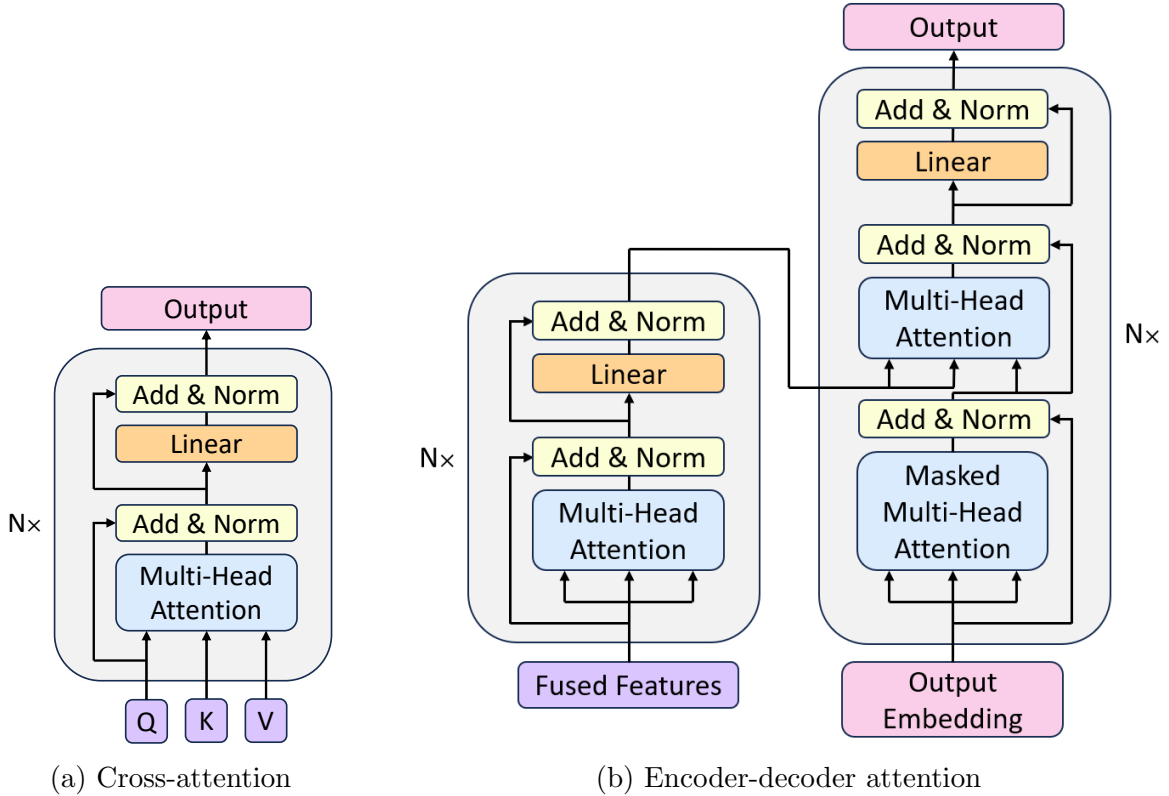


Figure 8: **Attention Block Details.** Self-attention is a special case of cross-attention with the same source.

serves as a crucial component within autoregressive models [14, 25, 62]. The autoregressive model operates by making predictions for a time series iteratively, one step at a time. To achieve this, it utilizes the previous step’s generated output as additional input for the subsequent prediction. This approach has demonstrated the capacity for mitigating accumulation of errors [13], which makes it desirable for longer-time predictions.

Figure 8 contains the transformer architecture details.

## D Experiment Setup

### D.1 Training

A standard cross-entropy loss  $\mathcal{L}_s$  is used for the symbolic outputs. Mean and standard deviation of the input sequence are computed, which are then used to normalize both the inputs and the labels. Mean squared error (in the normalized space)  $\mathcal{L}_d$  is used for the data predictions.

The data loss  $\mathcal{L}_d$  and symbol loss  $\mathcal{L}_s$  are combined to form the final loss function  $\mathcal{L} = \alpha\mathcal{L}_d + \beta\mathcal{L}_s$ , where the weights  $\alpha, \beta$  are hyperparameters. Unless otherwise specified, the models are trained using the AdamW optimizer for 30 epochs where each epoch is 2,000 steps. On a single NVIDIA GeForce RTX 4090 GPUs with 24 GB memory each, the training takes about 4.5 hours.

Table 8: **Model hyperparameters.** FFN means feedforward network.

Hidden dimension for attention	512	Hidden dimension for FFNs	2048
Number of attention heads	8	Fusion attention layers	8
Data encoder attention layers	2	Data decoder attention layers	8
Symbol encoder attention layers	4	Symbol decoder attention layers	8

Table 9: **Optimizer hyperparameters.**

Learning rate	$10^{-4}$	Weight decay	$10^{-4}$
Scheduler	Cosine	Warmup steps	10% of total steps
Batch size per GPU	512	Gradient norm clip	1.0
Data loss weight $\alpha$	5.0	Symbol loss weight $\beta$	1.0

## D.2 Hyperparameters

The PROSE model hyperparameters are summarized in Table 8, and the optimizer hyperparameters are summarized in Table 9.

For the FNO model in Section 5.2, we use 4 layers of standard 2d FNO to process the input data. The number of modes to keep in each dimension is set to 16, and the number of hidden channels is set to 64.

For the DeepONet model in Section 5.2, we employ the unstacked DeepONet architecture, consisting of a single trunk network and a single branch network. The input vectors are then passed through the branch network, producing an output with a basis dimension of  $p \times \dim_{\text{output}} = 20 \times 128$ . Simultaneously, the query point is processed through the trunk network, which also outputs a vector with the same dimension,  $p$ . Each element of output solution at the query point is obtained by taking the inner product of the outputs from the trunk net and the corresponding element of outputs from the branch net.

## E Additional Results

We present additional results in this section. Table 10 contains the errors per equation type as well as sample outputs.

Type		Parameters	Metrics		Worst Prediction		
			Rel- $L^2$	$R^2$			
Diffusion $u_t = q u_{xx}, t_f = 2$		$q_c = 3 \times 10^{-3}$	0.40%	1.000			
Porous Medium $u_t = (u^m)_{xx}, t_f = 0.1$		$m \in (2, 3, 4)$	0.35%	0.995			
Klein-Gordon $u_{tt} + (q^{(2)})^2 (q^{(1)})^4 u = (q^{(1)})^2 u_{xx}, t_f = 1$		$q_c^{(1)} = 1$ $q_c^{(2)} = 0.1$	0.53%	1.000			
Sine-Gordon $u_{tt} + q \sin(u) = u_{xx}, t_f = 1$		$q_c = 1$	0.58%	1.000			
Wave $u_{tt} = q u_{xx}, t_f = 1$		$q_c = 0.5$	0.33%	1.000			
Cahn-Hilliard $u_t + q^2 u_{xxxx} + c(u u_x)_x = 0, t_f = 0.5$		$q_c = 0.01$	0.38%	0.999			
Korteweg-De Vries $u_t + q^2 u_{xxx} + u u_x = 0, t_f = 1$		$q_c = 0.022$	0.94%	0.999			
Advection $u_t + q u_x = 0, t_f = 2$		$q_c = 0.5$	0.49%	1.000			
Fokker-Planck $u_t = D u_{xx} - \frac{1}{\gamma} (U(x)_x u)_x$ $t_f = 0.1, x_f = 2 \times 10^{-6}$		$U(x) = c \cos\left(\frac{x}{L}\right)$ $D = \frac{k_B T}{\gamma}$ $\gamma = 6\pi q^{(1)} r$	$c = 5 \times 10^{-21}$ $L = 10^{-7}$ $T = 300$ $r = 10^{-7}$ $q_c^{(1)} = 10^{-3}$	0.31%	0.998		
Diffusion Reaction $u_t = q^{(1)} u_{xx} + q^{(2)} R(u)$ $t_f = 2$		$R(u) = u$	$q_c^{(1)} = 3 \times 10^{-3}$ $q_c^{(2)} = 0.1$	0.44%	1.000		
		$R(u) = u(1-u)$		0.45%	0.999		
		$R(u) = u^2(1-u)$	$q_c^{(1)} = 3 \times 10^{-3}$ $q_c^{(2)} = 1$	0.54%	1.000		
		$R(u) = u^2(1-u)^2$		0.45%	1.000		
Conservation Law $u_t = -q^{(1)}(f(u))_x + \frac{q^{(2)}}{\pi} u_{xx}$ $t_f = 2$		Burgers': $f(u) = \frac{1}{2} u^2$		$q_c^{(1)} = 1$ $q_c^{(2)} = 0.01$	1.16%	0.999	
				$q_c^{(1)} = 1$ $q_c^{(2)} = 0$	2.45%	0.993	
		$f(u) = \frac{1}{3} u^3$		$q_c^{(1)} = 1$ $q_c^{(2)} = 0.01$	0.90%	0.999	
				$q_c^{(1)} = 1$ $q_c^{(2)} = 0$	2.19%	0.994	
		$f(u) = \sin(u)$		$q_c^{(1)} = 1$ $q_c^{(2)} = 0.01$	1.91%	0.996	
				$q_c^{(1)} = 1$ $q_c^{(2)} = 0$	4.26%	0.987	
		$f(u) = u$		$q_c^{(1)} = 1$ $q_c^{(2)} = 0.01$	2.14%	0.997	

Table 10: Results for PROSE-PDE 2-to-2 model with “Skeleton” inputs per type. The worst case prediction shows that the general trends and physical features are well-captured by the model.