Journal of Machine Learning **ISSN**: 2790-2048(e), 2790-203X(p)

A Multimodal PDE Foundation Model for Prediction and Scientific Text Descriptions

Elisa Negrini * [1], Yuxuan Liu[1], Liu Yang[2], Stanley J. Osher[1], and Hayden Schaeffer[1]

[1]Department of Mathematics, University of California Los Angeles, Los Angeles, CA 90095, USA.
[2]Department of Mathematics, National University of Singapore, Singapore 119077.

Abstract. Neural networks are one tool for approximating non-linear differential equations used in scientific computing tasks such as surrogate modeling, real-time predictions, and optimal control. PDE foundation models utilize neural networks to train approximations to multiple differential equations simultaneously and are thus a general purpose solver that can be adapted to downstream tasks. Current PDE foundation models focus on either learning general solution operators and/or the governing system of equations, and thus only handle numerical or symbolic modalities. However, real-world applications may require more flexible data modalities, e.g. text analysis or descriptive outputs. To address this gap, we propose a novel multimodal deep learning approach that leverages a transformer-based architecture to approximate solution operators for a wide variety of ODEs and PDEs. Our method integrates numerical inputs, such as equation parameters and initial conditions, with text descriptions of physical processes or system dynamics. This enables our model to handle settings where symbolic representations may be incomplete or unavailable. In addition to providing accurate numerical predictions, our approach generates interpretable scientific text descriptions, offering deeper insights into the underlying dynamics and solution properties. The numerical experiments show that our model provides accurate solutions for in-distribution data (with average relative error less than 3.3%) and out-of-distribution data (average relative error less than 7.8%) together with precise text descriptions (with correct descriptions generated 100% of times). In certain tests, the model is also shown to be capable of extrapolating solutions in time.

Keywords:
Multimodal AI,
Generative AI,
PDE foundation model,
Transformer-based operator learning,
Scientific text generation.

Article Info.:
Volume: X
Number: X
Pages: 1 - 28
Date: /2025
doi.org/10.4208/jml.250214

Article History:
Received: 14/02/2025
Accepted: 03/10/2025

Communicated by:
Lu Lu

1 Introduction

Neural networks have become increasingly important for solving non-linear differential equations, with applications in climate modeling, financial forecasting, biological systems analysis, and structural optimization (see for instance [8, 32, 36]). Their ability to model complex, non-linear relationships allows for efficient and accurate predictions for various scientific computing tasks such as surrogate modeling, real-time predictions, and optimal control. Previous work in deep learning for partial differential equations (PDE) have focused on learning either the solution operator, which maps input functions to their solutions, or the governing system of equations, which describes the constitutive model based on observations of state variables [6, 21, 22, 29, 35, 51]. These approaches, however, tackle one task at a time and are limited to the use of numerical data.

*Corresponding author. enegrini@math.ucla.edu

Building on the observation that families of differential equations frequently share fundamental characteristics, recent work has introduced transformer-based architectures to enable simultaneous encoding of various parametric differential equations [5,18,23,25,38, 46–48]. Although effective, these methods require structured input and output data, with vanilla in-context operator network (ICON) [46] focusing on numerical data and predicting operators and symbolic expressions using multimodal transformers (PROSE) utilizing numerical and symbolic data [25, 38]. In this work, we consider additional modalities as both inputs and outputs to the model. Often, one has access or would like to produce heuristic descriptions of the observed dynamics that are neither in symbolic nor numerical form, but instead come as text descriptions. For example, consider modeling the dynamics of a complex ecological system: the numerical inputs can include measured population levels, while text inputs could describe key processes such as predator-prey interactions or migration patterns. Similarly, in material science, numerical data could represent experimental results, while text inputs provide the governing equation or describe the experimental setup. By combining these modalities, the model may better capture the underlying rules and provide more accurate and contextually informed predictions. The use of mathematical formulae, text descriptions, and numerical values can provide a more robust development toward a PDE foundation model. Note that in fine-tune language models as multi-modal differential equation solvers (ICON-LM) [47] both textual and numerical prompts were provided as inputs; however, the model does not generate text descriptions since the outputs of ICON-LM are the numerical predictions.

Consider the following parametrized differential equation:

$$\begin{cases} \mathcal{F}\big(u(x,t;c)\big) = 0, & (x,t) \in \Omega \times [0,T], \\ \mathcal{B}\big(u(x,t;c)\big) = 0, & (x,t) \in \partial\Omega \times [0,T], \\ u(x,0;c) = \mathcal{G}(x;c), & x \in \Omega, \quad c \sim \mathcal{D}, \end{cases} \tag{1.1}$$

where $\mathcal{F}$ denotes the governing equation, $\mathcal{B}$ denotes boundary conditions. The initial condition $\mathcal{G}$ is a generating function, and c denotes the parameters that determine the initial conditions from distribution $\mathcal{D}$. The objective is to develop a single neural network model capable of approximating numerically and describing in text the solution operators for a range of governing equations $\mathcal{F}$ in (1.1), which can include ordinary differential equations (ODEs) and PDEs. For simplicity in experiments and data design, for the PDE dataset, we restrict our attention to periodic boundary conditions. However, we note that the model would be able to handle different boundary conditions simply by augmenting or extending the input by either providing these conditions as numerical values or analytic formulae. We leave the interesting and potentially more complex case of higher-dimensional PDEs for future work, as we expect this to also require a novel data processing as in [23].

The text input may contain either the equation to solve or a text description of the system. This is important in applications where one may only have a partial model or a description of the underlying process. The numerical data includes the equation's parameters and initial conditions. For instance, suppose we want to solve the heat equation with parameter $c = 0.003$ and initial condition $u(x,0) = u_0(x)$. The input to our model could be:

- A formal description: "The given equation is $u_t = cu_{xx}$, where $c = 0.003$ and $u(x, 0) = u_0(x)$."
- A more general description: "Solve the heat equation with parameter 0.003 and initial condition $u_0(x)$."

In our experiments, we primarily used the first type of input, where the equation is explicitly provided as a formula. However, given sufficient training examples, we expect the model to also perform well when only a descriptive name or general description of the equation is provided. In particular, the textual input description serves to disambiguate which parameters are being specified for the equation. For example, in equations with multiple parameters (such as Burgers' equation with advection strength and viscosity coefficients), the text input should explicitly name them (e.g., "advection strength = 1.0, viscosity coefficient = 0.01"), allowing the model to map the values to the correct components of the equation. With the input "The given equation is $u_t = cu_{xx}$ where $c = 0.003$ and $u(x, 0) = u_0(x)$," our model outputs:

1. **Numerical Predictions.** The solution to the PDE at user-defined query locations.
2. **Scientific Text Descriptions.** A sentence or more that describes the scientific properties of the equation or the solution. For instance:
   - Physical process modeled by the equation: "The heat equation is a parabolic PDE that models the spread of heat in a material over time."
   - Alternative numerical methods to approximate the solution: "The heat equation can be numerically solved using a combination of forward-time and central-space finite difference methods."

The generality of the generated text descriptions depends on the training set. For instance, descriptions of other properties of interest may be included in the training set, such as the long-term behavior of solutions, the presence of shocks or rarefactions, etc. We refer to Section 4.4, Table 4.4 for various examples of generated text. The text is collaboratively trained in the neural network with the goal of generalizing text descriptions to new systems.

As a representative example of our model's capabilities, in Section 4 we show that our model can determine whether the solution to a given conservation law from the conservation laws dataset will exhibit no shocks, shocks, or rarefactions. Specifically, on this dataset the numerical output achieves a test error of 1.41% (see Table 4.2). In Fig. 4.1, the last three rows illustrate that, when present, the locations and intensities of rarefactions and shocks are correctly identified. Finally, Table 4.4 shows that the text descriptions accurately capture shocks and rarefactions when present, with an F1 score greater than 0.94.

1.1 Main contributions

We proposes a novel multimodal framework that integrates numerical and textual input and output data for PDE foundation modeling. Our key contributions are as follows:

- **Multimodal Framework.** Our model integrates numerical inputs (e.g. initial conditions and parameters) with text descriptions of physical processes, allowing us to encode contextual information about the underlying dynamics. Moreover, by combining multimodal inputs with multimodal outputs (numerical solutions and text descriptions), our framework provides a novel, comprehensive, and interpretable modeling approach.

- **Custom Tokenization and Encoding.** Recognizing the potential limitations of GPT models in numerical tasks, we enhance the approach with a custom tokenizer to encode multimodal inputs into a unified token sequence. Textual data are tokenized using a pretrained GPT-2 backbone, while numerical data is encoded through a small multilayer perceptron (MLP). Continuous numerical encodings have also been proposed in [14], but without the use of an MLP. This encoding approach ensures compatibility between modalities and supports operator learning tasks sensitive to numerical accuracy.

- **Transformer-Based Operator Learning.** We employ a cross-attention-based transformer for numerical output generation, allowing the model to approximate solution operators efficiently. The numerical output decoder is designed to evaluate solutions at independent query points, ensuring scalability with respect to both time and space complexity.

- **Scientific Description Generation.** The model generates textual outputs using the GPT-2 backbone in an autoregressive manner. This enables the generation of descriptive and interpretable explanations of the system's behavior, including properties of the equation, of the solution, possible alternative methods to approximate the solution, etc.

2 Related works

2.1 Foundation models

Foundation models are large-scale, pre-trained models that can be fine-tuned for various downstream tasks in natural language processing [3, 42], computer vision [34], and other domains including robotics and biology [13, 50]. Despite their versatility, these models are not inherently suitable for number-sensitive tasks, such as scientific computing [44], PDE discovery [35], and time series forecasting [41], where high-precision solutions are critical. Early attempts have been made to adapt foundation models or similar transformer-based structures for scientific computing tasks. Some approaches utilize pre-training and fine-tuning techniques [7, 16], though these methods often require additional computational cost for downstream tasks. The ICON [5, 46–48] uses in-context learning to learn operators through example input-output pairs, which are few-shot learners and have demonstrated generalization capabilities for various differential equations [48]. Zero-shot PDE foundation models integrate additional information to aid the prediction process. ICON-LM [47] enables in-context learning with both text descriptions of governing physics and numerical data in the model inputs, but outputs are numerical only. In this way, text is used as a label to signal which PDE or task is needed. The PROSE framework [18, 23, 25, 38] is a multimodal approach that encodes numerical data along with symbolic representations of the PDE. In recent work, PROSE has been shown capable of extrapolation [38] and improved generalization through fine-tuning [39]. Other methods achieve zero- or few-shot generalization by embedding PDE structures within the network architecture [26, 49].

2.2 Multimodal machine learning

Multimodal machine learning focuses on models capable of integrating data from various modalities [27,37,40,45]. A key area of interest in this field is developing methods for information fusion across modalities, analyzing their interactions, and designing appropriate models and algorithms. For instance, in visual-language reasoning [20, 37, 40], combining visual data like images or videos and linguistic information such as captions or text descriptions [40] facilitates the development of models with enhanced multimodal understanding [20]. Similarly, AI robots utilize multimodal sensors, including cameras, radar, and ultrasound, to interpret their environments and make well-informed decisions [12,24]. Notably, the concept of a multimodal sentence was introduced in an embodied multimodal language model (PaLM-E) [11], where image and text can appear anywhere in the sentence and be processed in a flexible manner.

2.3 Transfer learning

Transfer learning has become a cornerstone of modern machine learning, enabling the adaptation of pre-trained models to specialized tasks with relatively limited data [31]. In the context of large language models (LLMs), such as GPT and bidirectional encoder representations from transformers (BERT), fine-tuning on domain-specific data allows the model to leverage its extensive pre-trained knowledge while focusing on the nuances of a target domain [9, 33]. For instance, fine-tuning GPT-2 or GPT-3 models on scientific texts has shown improved performance in tasks such as equation generation, technical summarization, and scientific question answering [3]. In our work, which follows the operator learning paradigm, we fine-tune an LLM with examples of numerical equations, symbolic representations, and descriptive text to enhance its ability to generate meaningful and contextually relevant outputs. The inclusion of this step in our framework, aligns well with recent advancements in multimodal tasks [15, 25, 38], demonstrating the importance of adapting pre-trained LLMs to specialized contexts.

3 Methodology

3.1 Model overview

The model consists of two main components: (1) pretrained GPT-2 backbone [33], and (2) cross-attention-based transformer for operator-style output evaluation similar to the

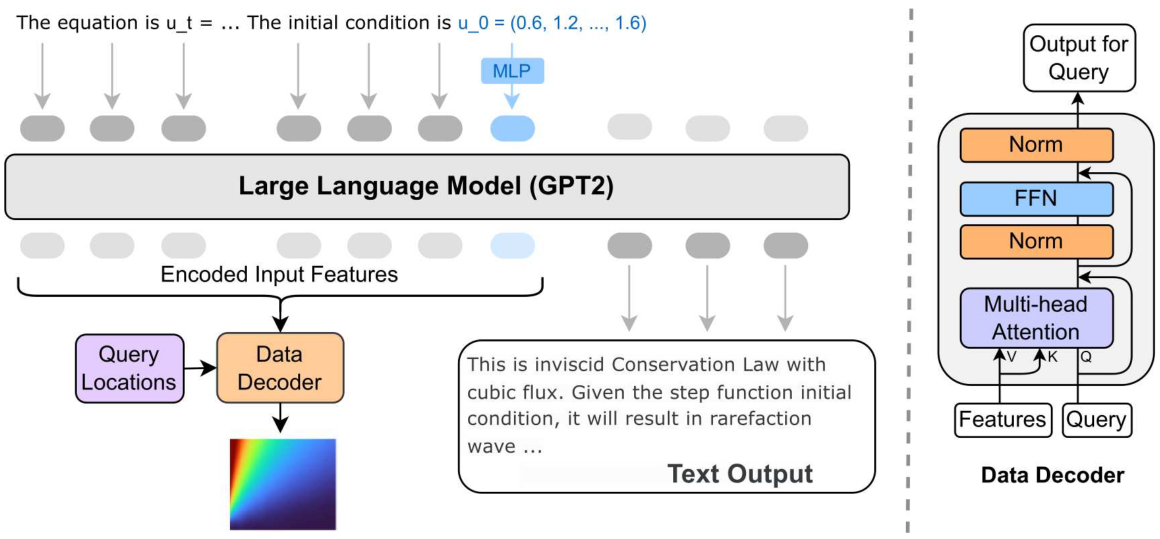
The equation is u_t = ... The initial condition is u_0 = (0.6, 1.2, ..., 1.6)
MLP
Large Language Model (GPT2)
Encoded Input Features
Query Locations
Data Decoder
This is inviscid Conservation Law with cubic flux. Given the step function initial condition, it will result in rarefaction wave ...
Text Output
Output for Query
Norm
FFN
Norm
Multi-head Attention
V
K
Q
Features
Query
Data Decoder

Figure 3.1: Model illustration. Our model processes multimodal input, where textual prompts describe equations, and numerical values represent initial conditions and parameters. A custom tokenizer encodes text using a GPT-2 tokenizer and numerical inputs via an MLP. The tokenized sequence is processed by an LLM backbone, followed by two decoding pathways: a transformer decoder for text generation and a data decoder with cross-attention to construct the operator.

one used in [25]. An illustration of the model is provided in Fig. 3.1. Given multimodal input data containing textual prompts describing the equation and numerical values representing initial conditions and parameters, our custom tokenizer first encodes the input data into a multimodal sequence of mixed tokens. The text input is encoded similarly to language models, where the GPT-2 tokenizer is used. For the numerical inputs, a small multilayer perceptron is used to encode the numerical data into feature vectors. This continuous encoding strategy similar to the one proposed in [14], demonstrates improved generalization, and is more suitable for number-sensitive tasks. The tokenized sequence is fed into the LLM backbone, which processes the input sequence and performs token mixing. Two different decoding methods are used to generate bi-modality output, and we discuss them in the following sections.

3.2 Transformers

Transformers are attention-based models that excel in tasks involving sequential data [43]. The attention mechanism [2] is the key component of transformers, which is used to weigh the importance of different elements in a sequence, allowing efficient parallelization and capture of long-range dependencies. This architecture has been highly successful in various domains, including natural language processing [33], computer vision [10], and more recently scientific computing [4,25,46]. It is also the building block of the GPT-2 backbone used in our model. Each layer in standard decoder-only transformers such as GPT-2 consists of two sub-layers: self-attention and feedforward layers. Given the input sequence

$X = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^{n\times d}$, each transformer layer processes the input as follows:

$$X' = \text{LayerNorm}(X), \tag{3.1}$$

$$Y = X + \text{MultiheadAttention}(X', X', X'), \tag{3.2}$$

$$\text{Output} = Y + \text{FeedForward}\big(\text{LayerNorm}(Y)\big) \in \mathbb{R}^{n\times d}. \tag{3.3}$$

The feedforward layer is a 2-layer multilayer perceptron, and the multihead attention mechanism (with h heads) is defined as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right) V, \tag{3.4}$$

$$\text{MultiheadAttention}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W^O, \tag{3.5}$$

where d_K is the number of columns of K, softmax is computed along each row, and each head is computed via

$$\text{head}_i = \text{Attention}\big(QW_i^Q, KW_i^K, VW_i^V\big) \in \mathbb{R}^{n\times \frac{d}{h}}. \tag{3.6}$$

Here, $Q, K,$ and V the input are the query, key, and value matrices of dimension $n \times d$, respectively (for self-attention, $Q = K = V = X$). $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d\times(d/h)}$ are the learned projection matrices for the i-th head, and $W^O \in \mathbb{R}^{d\times d}$ is the output projection matrix.

3.3 Operator evaluation

We generate numerical outputs representing equation solutions in the operator style. The Data Decoder in Fig. 3.1 employs a cross-attention mechanism to construct the operator, creating a connection between the LLM-processed input sequence and the output functions. The query locations, which represent the independent variables of these output functions, act as evaluation points. Notably, these query locations are independent of each other, meaning that evaluating the operator at one point does not influence its evaluation at another point. Consequently, the time and space complexity scale linearly with the number of query locations. Furthermore, since the evaluation points are decoupled from the network's encoding process, this approach resembles the principles underlying branch and trunk networks in DeepONet [28].

3.4 Autoregressive text generation

To generate text output, we use the standard autoregressive next-token prediction approach with the LLM backbone [43]. To generate the complete text output, we iteratively predict the next token in the sequence until a special end-of-sequence token is generated. At each step, the LLM generates the probability distribution for the next token, and we greedily select the token with the highest probability. Note that other decoding strategies such as beam search or sampling can be used to improve the quality of the generated text [19, 30]. During training time, to enable parallelization, we input the complete text sequence and explicitly mask out the future tokens for text generation and output tokens for Data Decoder.

3.5 Loss function

For the operator evaluation, we use the relative squared error $\mathcal{L}_n$, which makes learning more uniform among different families of equations with various data scales as shown in [17]. We also pad the equation solution to unify the data dimension so that the padded positions are not included in the loss calculation. For text generation, we use the standard cross-entropy loss $\mathcal{L}_t$ between the generated text and the ground truth text. The total loss is the sum of the two losses

$$\begin{aligned}\mathcal{L} &= \mathcal{L}\big((\boldsymbol{u},\boldsymbol{s}),(\hat{\boldsymbol{u}},\hat{\boldsymbol{s}})\big) = \alpha\mathcal{L}_n(\boldsymbol{u},\hat{\boldsymbol{u}}) + \beta\mathcal{L}_t(\boldsymbol{s},\hat{\boldsymbol{s}}) \\ &= \frac{\alpha}{B}\sum_{i=1}^{B}\frac{\|\boldsymbol{u}_i-\hat{\boldsymbol{u}}_i\|_2^2}{\|\boldsymbol{u}_i\|_2^2} + \frac{\beta}{B}\sum_{i=1}^{B}\text{CrossEntropy}(\boldsymbol{s}_i,\hat{\boldsymbol{s}}_i),\end{aligned}$$

where the weights α,β are hyperparameters to balance the two losses, i is the index for i-th element in the batch of size B, $\boldsymbol{u}$ is the ground truth numerical output, $\hat{\boldsymbol{u}}$ is the model numerical output, $\boldsymbol{s}$ is the text output ground truth, and $\hat{\boldsymbol{s}}$ is the model text output logits.

4 Results and discussion

This section presents the results of numerical experiments. In the first section, we describe the evaluation metrics, then we conduct four studies. First, we show the results on test data sampled from the same distribution as the training data. Second, we study the text generation performance for each equation class. Then we perform a study on out-of-distribution data. Finally, we test the ability of our network to extrapolate dynamics in time.

In all numerical experiments (Sections 4.3, 4.5, 4.6), the model receives a multimodal input, combining textual and numerical initial conditions and parameters, and is tasked with producing the numerical solution over the full time interval. While the model generates both numerical solutions and text descriptions, these sections only evaluate the model's numerical prediction capabilities. The text description output is evaluated separately in Section 4.4. In the first two numerical studies (in-distribution and out-of-distribution test data) we provide the initial condition (solution at time 0) and task the model to predict the solution on the time interval $[0,5]$. For the extrapolation in time study, the model is given its prediction of the solution at time 5 as the initial condition and asked to predict the solution on the longer time interval $[5,10]$.

The results in this section demonstrate that our model not only provides accurate solutions for equations where parameters are sampled within the same intervals as the training data, but also that it maintains a reasonable predictive accuracy for out-of-distribution (OOD) cases where parameters are sampled from larger intervals and, in some cases, is capable of extrapolating solutions in time. Furthermore, Section 4.4 demonstrates that our model is also able to produce accurate and consistent text descriptions.

4.1 Dataset overview

We generate a synthetic dataset consisting of 52 parametric differential equations, including both ODEs and PDEs of varying dimensions. The dataset is designed to cover a wide range of dynamics, including linear and non-linear systems, conservation laws, and reaction-diffusion equations. For each of the 52 parametric equations, we sample 100 parameters in the range $[Q - 0.1Q, Q + 0.1Q]$, where Q denotes the value of interest, resulting in a total of 5200 distinct equations. In addition to solution trajectories, each family of equations also comes with a set of text descriptions that describe the dynamics and properties of the system, generated using GPT-4 [1]. We enumerate all 52 parametric equations in Table 4.1. For more details about the dataset, we refer to Appendix A.

4.2 Evaluation metrics

We evaluate performance using a relative error for numerical outputs and the BERTScore for text outputs. These metrics are defined as follows:

1. Given a numerical prediction $\hat{\boldsymbol{u}}$ and ground truth solution $\boldsymbol{u}$, the relative error is defined as

$$\text{Relative Error} = \frac{||\hat{\boldsymbol{u}} - \boldsymbol{u}||}{||\boldsymbol{u}||},$$

   where $||\cdot||$ denotes the Euclidean norm.

   This metric measures the accuracy of the numerical solution generated by the model over the time interval compared to the true solution. The structure of $\boldsymbol{u}$ depends on the problem dimension:

   - For 1D ODEs, $\boldsymbol{u}$ is a vector containing solution values at discrete query time points.
   - For higher-dimensional ODEs, $\boldsymbol{u}$ is a matrix where each row corresponds to a different component of the solution, and each column represents a query time point.
   - For 1D PDEs, the solution $\boldsymbol{u}$ has spatial structure. We discretize the spatial domain into 128 points and treat each spatial point as an independent dimension. Thus, the solution is represented as a matrix of size $128\times$ (number of query time points), where each row corresponds to a specific spatial location, and each column represents a different query time.

2. BERTScore evaluates sentence similarity by comparing token-level representations using a pre-trained BERT model. Precision, recall, and F1 scores are calculated based on token alignments between two sentences. Precision measures the proportion of tokens in the generated sentence $\hat{x}$ that are relevant to the reference sentence x, indicating how much of the generated content matches the reference. Recall measures the proportion of tokens in the reference sentence x that are found in the generated sentence $\hat{x}$, reflecting how well the generated sentence captures the content of the

Table 4.1: Equations by type with corresponding indices. Each equation has parameters (such as a and b in the 1D ODE set) that are sampled during the data generation process.

Type	Equation	Index
1D ODE	$u_t = a\sin(2\pi t)u$	1
	$u_t = a\exp(-t) + b$	2
	$u_t = at^2\cos(u) + bu$	3
	$u_t = a\sin(\exp(-0.5t)\sin(3\cdot t)) + bu$	4
	$u_t = at\sin(u)$	5
2D ODE	Van der Pol	8
	Lotka-Volterra	9
	FitzHugh-Nagumo	10
	Brusselator	11
	Duffing	12
3D ODE	SIR model	6
	Neural dynamics	7
PDE	Heat	13
	Porous medium	14
	Klein Gordon	15
	Sine Gordon	16
	Cahn Hilliard	17
	Korteweg-de Vries	18
	Advection	19
	Wave	20
	Diffusion-reaction logistic	21
	Diffusion-reaction linear	22
	Diffusion-reaction bistable	23
	Diffusion-reaction square logistic	24
	Fokker-Plank	34
Conservation laws	Burgers	25
	Inviscid Burgers	26
	Conservation law linear flux	27
	Conservation law cubic flux	28
	Inviscid conservation law cubic flux	29
	Conservation law sine flux	30
	Inviscid conservation law sine flux	31
	Conservation law cosine flux	32
	Inviscid conservation law cosine flux	33
	Burgers-inviscid conservation law cosine flux with one shock	35-43
	Burgers-inviscid conservation law cosine flux with rarefaction	44-52

reference. The F1 score is the harmonic mean of precision and recall, balancing relevance and coverage. An F1 score of 1 indicates perfect precision and recall, while lower scores suggest imbalances between the two. Precision, recall, and F1-score are computed as follows:

$$\text{BERTScore-R} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} x_i^\top \hat{x}_j,$$

$$\text{BERTScore-P} = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} x_i^\top \hat{x}_j,$$

$$\text{BERTScore-F1} = \frac{2 \cdot \text{BERTScore-P} \cdot \text{BERTScore-R}}{\text{BERTScore-P} + \text{BERTScore-R}},$$

where x_i and $\hat{x}_j$ are the token embeddings from the reference x and candidate $\hat{x}$, respectively, and $x_i^\top \hat{x}_j$ represents their dot product.

4.3 Numerical predictions on test data

In this section, we present numerical results on test data sampled from the same distribution as the training data. As detailed in Section 4.1, to generate the training data, we uniformly sample the parameters of each differential equation within the range $[Q - 0.1Q, Q + 0.1Q]$, where Q denotes the parameter value of interest. This range corresponds to a 10% relative variation around each parameter's nominal value. In this section, we evaluate the in-distribution performance of our method, i.e. its performance on test data where the parameters are also uniformly sampled from the range $[Q - 0.1Q, Q + 0.1Q]$.

Table 4.2 shows the results for each equation class. We can see that in all cases the low relative error is achieved (less then 5.4%) with errors especially low for the PDEs and Conservation laws classes (less then 1.9%). This discrepancy between ODE and PDE sets can be explained by the fact that low-dimensional systems often have simpler dynamics, making them more sensitive to perturbations and modeling inaccuracies. This sensitivity can make small deviations more pronounced and result in larger relative errors. Fig. 4.1 shows one prediction example per class. As explained above, for testing the only numerical data provided to the model was the initial condition and the equation parameters in the multimodal input sentence and the model was tasked to predict the solution on $[0, 5]$. From the figure, we see that the prediction is almost indistinguishable from the ground truth. For the PDE and Conservation laws examples the largest error is observed where

Table 4.2: Relative errors on test data per class.

Class	Relative test error (%)
1D ODE	3.00%
2D ODE	5.36%
3D ODE	4.43%
PDE	1.85%
Conservation laws	1.41%
Total average	3.21%

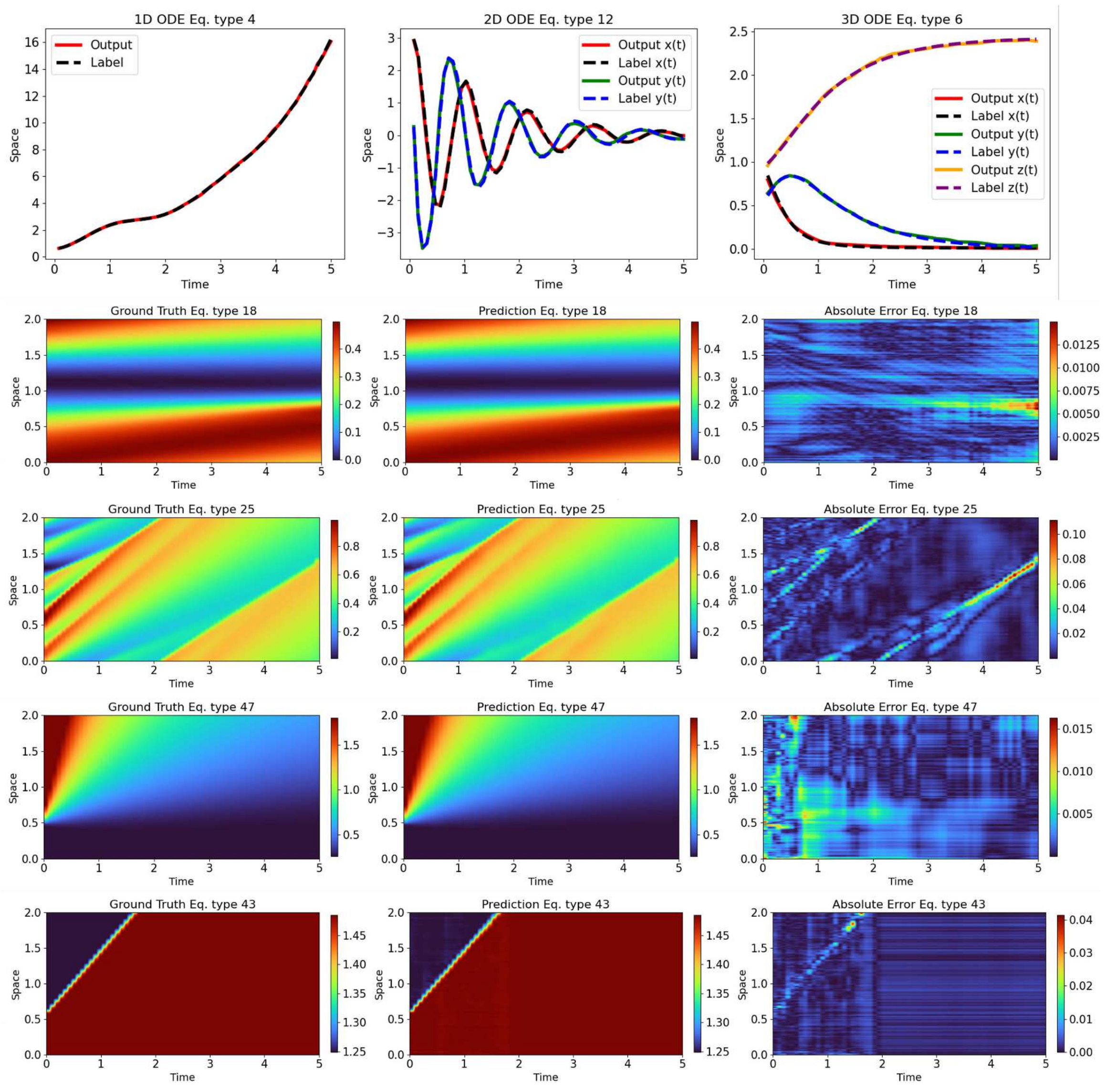

Figure 4.1: Comparison of outputs: For PDE examples, we show from left to right the ground truth solution, the predicted solution, and the absolute difference. First row: from left to right 1D ODE (index 4), 2D ODE (index 12, Duffing system), 3D ODE (index 7, Neural dynamics). Second row: PDE (index 18, Korteweg De Vries equation). Third row: Conservation law (index 25, Burgers' equation). Fourth row: Conservation law with rarefaction (index 47, Inviscid conservation law cubic flux); Fifth row: Conservation law with shock (index 43, Inviscid conservation law cosine flux).

there is a sharp transition in the dynamics. However, in all cases, the main features of the solutions are correctly predicted. Notably, in the case of conservation laws with rarefaction and shocks (rows 4 and 5 of the figure), the location of both the rarefaction and shock are correctly identified.

4.4 Scientific text description generation

Each equation in our dataset was associated with 50 different text descriptions. During training, given the equation in text form and the initial condition as numerical data, the model produces a text description. In this work, the text descriptions for training were generated using GPT-4 and described multiple properties of the equation to be solved. Some descriptions focus on the equation and its properties (such as order, linearity, and main equation terms), some focus on classical numerical methods which may be used as an alternative to produce an approximate solution, and some focus on the natural dynamics that the equation describe (for example the Lotka Volterra system describes the interactions between prey and predators in an ecosystem). For evaluation, we accept as "correct" any description that is consistent with the corresponding equation.

To evaluate the accuracy of generated text descriptions, we use BERTScore to measure sentence similarity. Specifically, for a given input, we generate an output text description and compare it with a randomly selected text description from the set of 50 associated with the equation. The results of this comparison on test data for each class are shown in Table 4.3. From the table, we see that the average BERTscore is high (greater then 0.935) for all classes. Note that the BERTScore is slightly below a perfect match because of the inherent variability in the text descriptions. When we randomly select a description for comparison, it may focus on a different property of the equation than the one described by the generated output. For example, one description may emphasize the order of the equation, while another focuses on a classical numerical method used to approximate its solution. Although both descriptions are consistent and correct for the given equation, their semantic similarity would be low due to the differences in focus. This discrepancy leads to slightly reduced BERTScores, even when the generated descriptions are valid. For 100 test examples, we manually verified that the text descriptions were consistent with the provided multimodal input and found them to be accurate 100% of the time.

Table 4.4 shows the input data, example description for that class, generated text, and corresponding BERTscore-F1 for the numerical results from Section 4.3, Fig. 4.1. We see that in all cases the generated description is consistent with the input data, i.e. it correctly describes the equation or the properties of the solution. However, the BERTScore-F1 is not perfect because of small semantic discrepancies between the example and generated descriptions. For example in the 1D ODE case the example description mentions "a decaying exponential term" while the generated text mentions "a time-varying control term".

Table 4.3: BERTScore results per class.

Class	BERTScore – Precision	BERTScore – Recall	BERTScore – F1
1D ODE	0.919	0.915	0.917
2D ODE	0.950	0.950	0.950
3D ODE	0.918	0.915	0.916
PDE	0.947	0.949	0.948
Conservation laws	0.952	0.952	0.952
Total average	0.937	0.936	0.937

Table 4.4: Example and generated text descriptions and BERTScore-F1 similarity with corresponding input data for different equations.

Class (index)	Input data	Example description	Generated text	F1 Score
1D ODE (index 4)	The ODE is $u_t = a\sin(e^{-0.5t})\sin(3t)) + bu$ we have initial data $u = \dots$ and coefficients $[a, b] = \dots$	This first-order linear equation includes a sine function affected by a decaying exponential term and is linear in u.	The ODE is characterized by a first-order linear term with a sine function influenced by a time-varying control term.	0.911
2D ODE (index 12)	The Duffing system is $x_t = y$, $y_t = -\delta y - \alpha x - \beta x^3$. The initial data is $[x, y] = \dots$, and the parameters are $[\alpha, \beta, \delta] = \dots$	The Duffing system, with α, β, and δ, shows how cubic stiffness and damping lead to non-linear oscillations in x and y.	In the Duffing system, parameters α, β, and δ illustrate how cubic stiffness and damping lead to non-linear oscillatory behavior in x and y.	0.977
3D ODE (index 7)	The Neural dynamics system is $E_t = \alpha E - \beta EI - \gamma E + 0.01\sin(t)$, $I_t = \delta E - \epsilon I, \quad H_t = \theta I - \phi H$. We have initial data $[E, I, T] = \dots$, parameters $[\alpha, \beta, \gamma, \delta] = \dots$	This model describes how sinusoidal input influences neural excitatory and inhibitory balance.	This model describes how excitatory and inhibitory neural signals evolve under sinusoidal stimulation.	0.960
PDE (index 18)	The equation is $u_t + q^2 u_{xxx} + u\,u_x = 0$ with observed initial data $u = \dots$ and coefficient $q = \dots$	The equation of Korteweg–de Vries captures wave propagation in shallow channels, influenced by nonlinearity and dispersion.	The equation of Korteweg–de Vries describes the behavior of solitons in shallow water, balancing nonlinear and dispersive forces.	0.945
Conservation law (index 25)	The equation is $u_t = -q_1(f(u))_x + \frac{q_2}{\pi} u_{xx}\ f(u) = 1/2\,u^2$. We have initial condition $= \dots$ and coefficients $[q_1, q_2] = \dots$	The viscous Burgers' equation, with sinusoidal initial conditions, models fluid dynamics that stay smooth without shocks due to viscosity.	The viscous Burgers' equation models fluid dynamics initiated by sinusoidal waves, ensuring no shocks form due to viscosity.	0.942
Conservation law rarefaction (index 48)	The equation is $u_t = -k(f(u))_x$, where $f(u) = 1/3\,u^3$. Initial condition $= \dots$ and coefficient $k = \dots$	The solution of the inviscid conservation law with cubic flux, starting from step function initial conditions, demonstrates a rarefaction wave forming.	With step function initial conditions, the inviscid conservation law with cubic flux models the fluid flow where a rarefaction wave forms.	0.941
Conservation law shock (index 43)	The equation is $u_t = -k(f(u))_x$, where $f(u) = \cos(u)$. Initial condition $= \dots$ and coefficient $k = \dots$	The inviscid conservation law with cosine flux, initialized from a step function, leads to fluid dynamics featuring one shock.	The inviscid conservation law with cosine flux, starting with a step function, ensures fluid dynamics include one shock.	0.963

Both of these descriptions are consistent and correct with respect to the input data ("The ODE is $u_t = a\sin(\exp(-0.5t)\sin(3t)) + bu$ we have initial data $u = \dots$ and coefficients $[a, b] = \dots$"), but semantically they are different, hence the reduced BERTScore of 0.911.

An interesting scientific application of our method can be seen when we apply it to conservation laws (see the last 3 rows of Table 4.4). In that case, we are interested in generating text descriptions that can characterize the solution behavior and determine whether shocks or rarefactions are likely to form based solely on the input data. This capability of our method is particularly significant as being able to predict whether shocks or rarefactions will form directly from input data is crucial for understanding and controlling systems governed by conservation laws, especially when dealing with complex, non-linear dynamics arising for example in fluid dynamics, traffic flow, and material science applications. This example shows that our multimodal LLM method is able to leverage physical knowledge encoded in equation structures (text input data) alongside numerical data (such as initial conditions and parameters) to produce informed conclusions (determining the presence of shocks or rarefactions). Finally, the ability of our model to produce as output simultaneously a numerical solution (see Fig. 4.1) and a text description (see Table 4.4) allows for a deeper understanding of the problem and more interpretable results.

These examples demonstrated the ability of our model to produce any description coherent with the input data without requiring the description to focus on a specific property of the equation or of the solution. A simple modification of this model could be to include in the input data a short text prompt that specifies what property of the problem the output text description should focus on. For example, we could require a description of the long-time behavior of the solution, a possible discretization of the equation, or if the given equation can be used to model real-world dynamics. We leave this as future work as this would require a more general example description set.

4.5 Out-of-distribution testing performance

In the previous section, the train and test datasets were generated with parameters uniformly sampled from the range $[Q - 0.1Q, Q + 0.1Q]$ where Q was the quantity of interest. In this section, we test our algorithm on out-of-distribution data, specifically, the test data parameters are sampled from $[Q - 0.2Q, Q + 0.2Q]$ (20% relative range) and from $[Q - 0.3Q, Q + 0.3Q]$ (30% relative range). Table 4.5 shows the test errors respectively for 10% relative range (in-distribution test data), 20%, and 30% relative range (out-of-distribution test data). As expected, we observe an increase in the relative error as we increase the parameter's relative range; however, the error remains low for most equation classes. The average error is less then 3.3% for the 10% range, less then 7.8% for the 20% range, and less then 12.0% for the 30% range.

Table 4.5: Relative errors on out-of-distribution test data. From left to right, results on 10% relative range (in-distribution data), and results from 20% and 30% relative range for each equation class.

Class	10% Relative range (training)	20% Relative range	30% Relative range
1D ODE	3.00%	7.19%	12.56%
2D ODE	5.36%	15.15%	25.71%
3D ODE	4.43%	6.46%	8.86%
PDE	1.85%	3.56%	5.82%
Conservation laws	1.41%	3.54%	6.66%
Total average	3.21%	7.78%	11.92%

We note that the 2D ODE class exhibits the largest increase in error as we expand the parameter range. This behavior is due to the sensitivity of these systems to changes in their parameters. Many of these equations, such as the Van der Pol oscillator, Lotka-Volterra model, FitzHugh-Nagumo model, Brusselator, and Duffing oscillator, exhibit highly nonlinear and often oscillatory dynamics. In such systems, even small deviations in parameter values can result in significant changes in the amplitude, frequency, or overall trajectory of the solutions leading to dynamics that deviate substantially from those seen during training. For example, for the Duffing oscillator small parameter shifts can move the system between periodic and chaotic behaviors while for Van der Pol small changes in parameters can shift the system between weakly and strongly oscillatory regimes.

As the relative parameter range increases, we observe that the error roughly doubles. Since our model relies solely on the initial condition as numerical input, it must infer the entire trajectory based on limited information, which can amplify errors when the system's behavior deviates significantly from the training data. A possible way to mitigate this issue is to provide the model with additional time steps of the solution as input. Including more time steps offers the model a richer context about the dynamics, allowing it to identify trends and patterns that remain consistent even as parameters vary. This additional information can stabilize and regularize the learning process, reducing the model's reliance on extrapolation from the initial condition and improving its ability to adapt to unseen parameter values [25]. We leave the addition of multiple time steps as input as a future work.

4.6 Extrapolation in time

In this section, we study the ability of our method to extrapolate the dynamics in time. As explained in Section 4.1, the model is trained on data generated on the time interval $[0, 5]$. In the previous sections, the trained model received as numerical input the solution at time $t = 0$ (the initial condition) and was tasked with producing the solution over the entire time interval $[0, 5]$. In contrast, in this section, we investigate the model's capacity for temporal extrapolation by providing, as the initial condition, its own approximation of the solution at time $t = 5$. The model is then tasked with generating the solution on the time interval $[5, 10]$. Note that we only show extrapolation results for certain classes of equations where the solution at time $t = 5$ is similar to the initial condition used for training. This is because if the solution at $t = 5$ differs significantly from the training data the new input may be an OOD example.

Table 4.6 shows extrapolation results for heat equation, advection equation, diffusion-reaction equation, inviscid conservation law with cubic flux and conservation law with sine flux. For each class, we sample 70 different parameters and initial conditions and report the average relative error in the numerical solutions over the time interval $[5, 10]$. We see that the extrapolation error is less then 13% for most equation classes showing the ability of our model to extrapolate in time in most cases. For the advection equation class, the error is quite high (around 30%) due to the nature of the solution, which often includes sharp changes. While the general shape of the solution can often be predicted, if the initial approximation at $t = 5$ is slightly shifted or inaccurate, these sharp transitions

Table 4.6: Relative errors for extrapolation in time for different equations classes. The model is given as numerical input its approximation of the solution at time $t = 5$ and tasked to produce the solution on the time interval $[5, 10]$.

Equation	Relative error (%)
Heat equation	8.65%
Advection equation	29.3%
Diffusion-reaction square logistic	9.08%
Inviscid conservation law cubic flux	7.94%
Conservation law sine flux	12.9%

cause a significant increase in the error on the time interval $[5, 10]$ (see Fig. 4.2 row 2 for an example of an advection equation extrapolated solution with sharp intensity changes).

Fig. 4.2 shows one example of extrapolated solution per class on the time interval $[5, 10]$. By comparing the ground truth (first column of the figure) with the prediction (second column of the figure) we can see that in all cases the general dynamic is correctly extrapolated. The absolute error (last column in the figure) indicates that the largest errors occurs where the solution exhibits sharp changes in intensity. This is expected, as even a minor deviation in the solution at $t = 5$ can lead to a shift in the prediction, resulting in a large absolute difference. This occurs even when the average error remains small and the main features of the dynamics are accurately captured.

We note explicitly that larger errors for extrapolation in time are expected. This setting presents two main challenges for our model. First, it relies solely on the initial condition to generate the entire trajectory. Second, this initial condition is not exact, but rather the model's own prediction at $t = 5$. As such, it already contains some degree of error or noise. This compounds the difficulty, as the model must now predict future states based on a single data point that is itself inexact. One possible way to mitigate this issue is to provide the model with the solution at multiple time steps to give a broader temporal context. However, when these additional steps are themselves generated by the model, they may still carry accumulated inaccuracies. Despite this, incorporating multiple past predictions could help the model recognize trends and reduce error growth. This would be similar to the numerical input used in [25].

4.7 Comparison with other methods

Comparison with Standard Operator Learning. Popular operator learning methods, such as Fourier neural operators (FNO) [21] and DeepONet [29], assume a fixed underlying equation across training samples and learn mappings from function spaces to function spaces using only numerical inputs. In contrast, our network encodes multiple equations simultaneously, with only initial conditions and equation parameters provided as numerical data (while equations are encoded in the text input). This yields a problem ill-posed for standard operator learning methods, which would have to infer solely from the numerical input both the solution operator and governing equation (including unknown forces), making a fair comparison not feasible. As a baseline test, [25, Table 5] compared FNO and DeepONet with the PROSE model applied to ODE examples, where historical

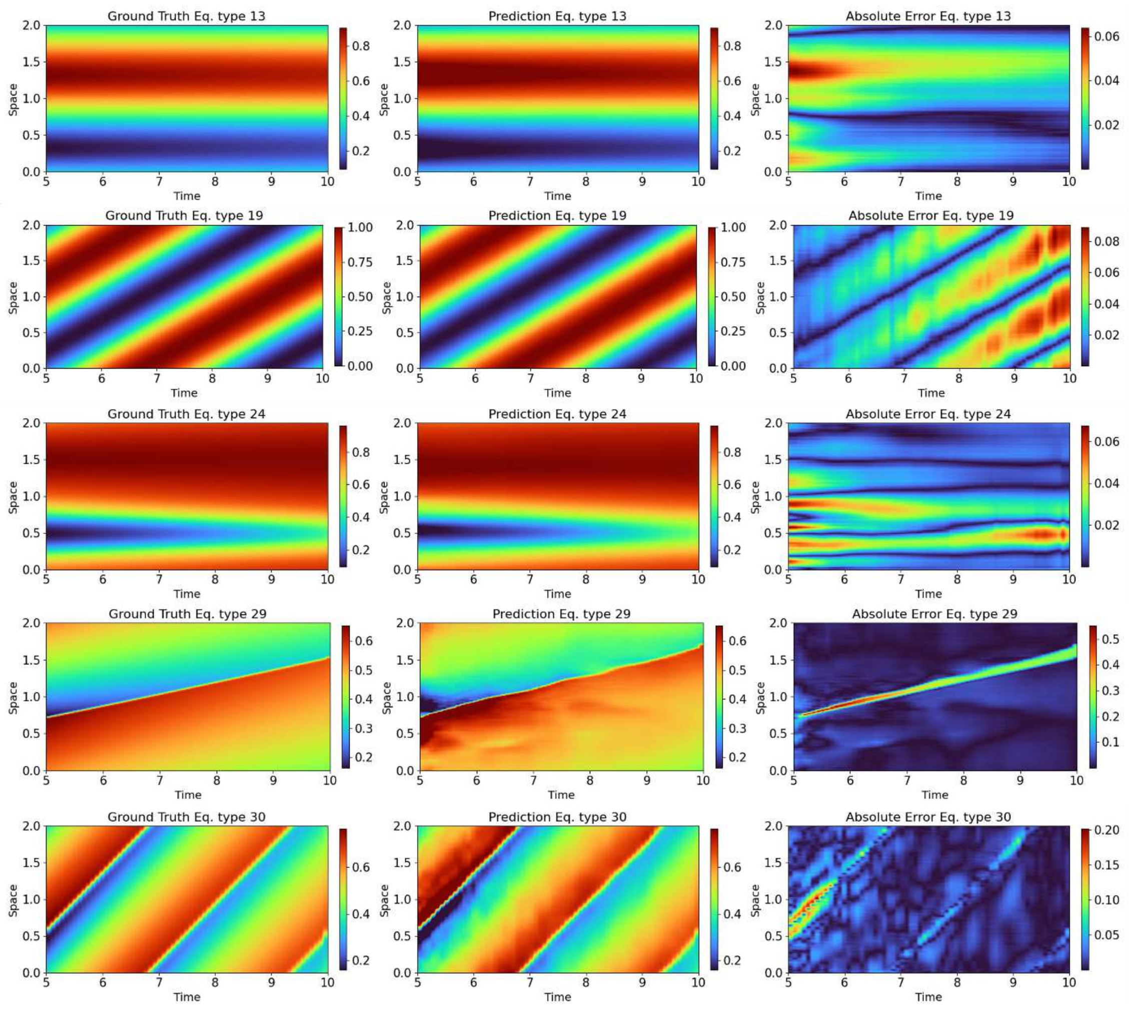

Figure 4.2: Example of extrapolation in time. The model is trained on data generated for $t \in [0,5]$ and tasked to predict the solution for $t \in [5,10]$. From left to right, ground truth solution, predicted solution, absolute difference. First row: Heat equation (index 13), Second row: Advection equation (index 19). Third row: Diffusion-reaction square logistic (index 24). Fourth row: Inviscid conservation law cubic flux (index 29). Fifth row: Conservation law sine flux (index 30).

context is provided, i.e. multiple input snapshots of the solution are available instead of only the initial condition. Even with this additional information, DeepONet and FNO produced relative errors respectively of 8.51% and 9.40%. Similarly, in [38, Table 5], FNO and DeepONet are compared to the PROSE-PDE model on the same PDE dataset used in this paper (noting that [38] uses multiple input snapshots, while we only use the initial condition). For the PDE examples, the relative errors for DeepONet and FNO were respectively 32.07% and 34.86%, while for our model the relative error was 1.85%. Finally in Table 4.7 we summarize the differences and similarities between our method and other state-of-the-art methods in terms of model capabilities, inputs and outputs.

Comparison with ICON-LM and PROSE. While our work shares some high-level goals with ICON-LM [47] and PROSE [25], the underlying methodology is fundamentally different in key aspects. ICON-LM relies on in-context learning by providing the model with multiple input-output pairs (e.g. initial conditions and their corresponding solutions) during inference. In contrast, our model uses a fixed architecture trained across systems and does not rely on in-context examples at inference time, making a direct comparison with ICON-LM not meaningful. Similarly, PROSE encodes PDEs using symbolic forms in Polish notation and does not use natural language text as input. Our model, on the other hand, takes textual descriptions, including analytic LaTeX formulas, as input, which allows us to leverage pretrained language models such as GPT-2 for encoding. Rather than training a text encoder entirely from scratch, as done in PROSE, we use GPT-2's pretrained representations of language and mathematical tokens and only fine-tune it on our domain-specific data using a small learning rate. This design provides flexibility in how equations and parameters can be described, since GPT-2 already has a rich prior over natural and formal language, and efficiency, as it eliminates the need to retrain the entire model end-to-end. In contrast, PROSE requires training the full model from scratch for symbolic reasoning over equations. Furthermore, a key novelty of our method lies in its ability to generate textual output descriptions of the dynamics, which neither ICON-LM nor PROSE are designed to do. These similarities and differences are summarized in Table 4.7. Although, as explained above, a comparison of our method with PROSE may not be fair, we provide a comparison of the numerical accuracy obtained by our method and PROSE-PDE [38] on the PDE dataset when only the initial condition is provided as numerical data. Specifically, we use the PROSE-PDE 2-to-1 model with the default hyperparameters (introduced in [38, Section 5.1]), where the symbol decoder is omitted, thus the model only produces the numerical solution. As input for the PROSE-PDE 2-to-1 model we provide the full equation in symbolic form (which includes the equation coefficients) and the initial condition. For our model the input provided contains the equation in text format, the initial condition and equation coefficients as in our previous examples. We then compare the nu-

Table 4.7: Comparison of methods by input/output types and multi-operator learning capability.

Method	Input type	Output type	Multi-operator learning
Our method	numeric text symbolic (LaTeX)	numeric text symbolic (LaTeX)	✓
PROSE	numeric symbolic (Polish notation)	numeric symbolic (Polish notation)	✓
ICON-LM	numeric text in-context	numeric	✓
FNO, DeepONet	numeric	numeric	✗

merical outputs of these two models on the time interval $[0, 5]$. For this data our method produced an error of 1.35% while PROSE-PDE produced a relative error of 2.27%. We believe this performance difference is due to two main factors. First, the use in our approach of pre-trained GPT-2: the textual input is encoded using pretrained GPT-2 which already contains rich representations of natural language and mathematical tokens. We only fine-tune it (using a small learning rate) for our specific setting. This transfer learning reduces the burden of learning text representations from scratch and allows the model to generalize more effectively to diverse PDE descriptions. In contrast, PROSE-PDE trains its symbol encoder from scratch using Polish notation, which requires learning both the syntactic and semantic structure of the symbolic representation without any pretrained priors. Second, PROSE-PDE has fewer parameters overall (approximately 71.5 million) than our model (approximately 150 million), which limits its ability to capture both textual and numerical dependencies with the same accuracy.

Comparison with ChatGPT. While large language models like ChatGPT are capable of generating textual descriptions from prompts such as the name of a PDE or its equation, they lack the ability to combine the text input to the numerical context which is necessary to produce accurate and physically meaningful classifications of solution behavior. Our method differs significantly from language models such as ChatGPT in that it combines both textual input and numerical data to generate output descriptions. This multimodal approach connects the textual descriptions with the observed data, which is particularly valuable in cases where the same equation name alone may refer to multiple dynamics. For instance, in our experiments on conservation laws, for the same equation, different initial conditions may result in different solution behaviors such as shock formation or rarefaction patterns so a simple strategy like providing the equation name as a prompt to ChatGPT would not provide an accurate dynamics description. Moreover, because of the tokenization used by these methods, even adding numerical data to the ChatGPT prompt would not result in the same accuracy as our model. To explore the ability of ChatGPT (Version 3.5) to produce meaningful textual descriptions we manually provided it with the same input as our model and asked to classify the equation, the initial condition and determine if no shocks, shocks or rarefactions formed in the solution. In particular, an example of a prompt given (omitting for brevity the numerical data) is:

```
The equation is ut = -q1(f(u))x + q2/pi uxx where f(u) = 1/3 u^3.
Initial condition are = [...] and coefficients [...].
With one sentence describe the equation, the initial condition and
if there are shocks or rarefactions in the solution.
```

Table 4.8 shows the results of this comparison on 8 different conservation laws inputs. The misclassifications are highlighted in red. For these inputs our model was correct 100% of the times in identifying the equation, initial condition and solution behavior. ChatGPT gave completely correct answers for 2 out of 8 examples, it misclassified the equation in 1 out of 8 examples and incorrectly classified solution behavior in 6 out of 8 examples.

Table 4.8: Comparison table with input equations, true description, ChatGPT-generated answer, and our output.

Input	True classification	ChatGPT output	Our model's output
The equation is $u_t = -q_1(f(u))_x + q_2\pi u_{xx}$, where $f(u) = 1/2u^2$, initial data $u = \ldots$, coefficients $[q_1, q_2] = \ldots$	Viscous Burgers'. Sinusoidal initial conditions. No shocks or rarefactions.	This is the viscous Burgers equation with initial conditions smooth and decreasing, resulting in a rarefaction wave.	The viscous Burgers' equation, with sinusoidal initial conditions, ensures fluid dynamics remain smooth and free of shocks due to viscosity.
The equation is $u_t = -k(f(u))_x$, where $f(u) = 1/2u^2$, initial data $u = \ldots$, coefficients $[k] = \ldots$	Inviscid Burgers. Step function initial condition resulting in one shock.	This is the inviscid Burgers equation with a discontinuous initial condition, which leads to the formation of a shock.	The inviscid Burgers' equation captures fluid behavior with a step function initial condition, resulting in one shock.
The equation is $u_t = -q_1(f(u))_x + q_2\pi u_{xx}$, where $f(u) = u$, initial data $u = \ldots$, coefficients $[q_1, q_2] = \ldots$	Viscous conservation law with linear flux. Step function initial condition which leads to rarefaction wave.	This is a viscous Burgers' equation with a jump in the initial condition, which results in a shock.	The viscous conservation law with linear flux captures fluid dynamics, where a rarefaction wave forms due to step function initial conditions.
The equation is $u_t = -q_1(f(u))_x + q_2\pi u_{xx}$, where $f(u) = \sin(u)$, initial data $u = \ldots$, coefficients $[q_1, q_2] = \ldots$	Viscous conservation law with sine flux. Step function initial condition which leads to rarefaction wave.	This is a nonlinear convection-diffusion equation with sine nonlinearity, initialized with a nearly constant step-like profile, where a shock forms.	The Viscous conservation law with sine flux and step function initial conditions models the gradual development of a rarefaction wave in the fluid.
The equation is $u_t = -q_1(f(u))_x + q_2\pi u_{xx}$, where $f(u) = 1/3u^3$, initial data $u = \ldots$, coefficients $[q_1, q_2] = \ldots$	Viscous conservation law with cubic flux. Sinusoidal initial conditions, no shocks.	The nonlinear convection-diffusion equation with smooth, gradually varying initial conditions and given coefficients produces a solution exhibiting rarefaction waves.	The viscous conservation law with cubic flux describes fluid behavior from sinusoidal initial conditions, ensuring no shock formation.
The equation is $u_t = -q_1(f(u))_x + q_2\pi u_{xx}$, where $f(u) = 1/2u^2$, initial data $u = \ldots$, coefficients $[q_1, q_2] = \ldots$	Viscous Burger's. Step function initial condition leading to a shock.	The nonlinear convection-diffusion equation with a piecewise constant initial condition featuring a sharp jump produces a solution where shocks form.	The viscous Burgers' equation, starting with a step function, describes fluid behavior with one shock forming due to viscosity.
The equation is $u_t = -q_1(f(u))_x + q_2\pi u_{xx}$, where $f(u) = \cos(u)$, initial data $u = \ldots$, coefficients $[q_1, q_2] = \ldots$	Viscous conservation law with cosine flux. Step function initial condition leading to rarefaction.	The nonlinear convection-diffusion equation with a piecewise constant initial condition that jumps sharply from a low to a high value and given coefficients produces a solution where shocks form.	The viscous conservation law with cosine flux and step function initial conditions models fluid dynamics where a rarefaction wave forms.
The equation is $u_t = -k(f(u))_x$, where $f(u) = \sin(u)$, initial data $u = \ldots$, coefficients $[k] = \ldots$	Inviscid conservation law with sine flux. Sinusoidal initial conditions. No shocks.	This is a convection-diffusion equation with flux sin(u). The initial condition is smooth and oscillatory. It will develop rarefactions.	The inviscid conservation law with sine flux models smooth fluid motion starting from sinusoidal waves, avoiding shocks.

5 Conclusion

This work introduces a multimodal transformer-based framework for operator learning that integrates numerical and textual data to approximate solution operators for a diverse class of ODEs and PDEs. By incorporating both types of data, our model bridges the gap between numerical predictions and scientific interpretability, making it well-suited for applications where numerical and textual inputs/outputs are required. The framework's ability to output both numerical solutions at query locations and scientifically meaningful text descriptions of system dynamics represents a step toward interpretable and comprehensive PDE foundation modeling.

Our experiments demonstrated that the proposed approach is able to produce accurate solutions for in-distribution data, with average relative error less than 3.3%, and out-of-distribution data, with average relative error less than 7.8%, showcasing its robustness and generalization capabilities. Notably, the model can also extrapolate solutions over extended time intervals. Finally, the generated text descriptions are both accurate and informative, with coherent descriptions generated 100% of times, covering diverse aspects of the equations and their solutions, including physical interpretations and alternative numerical methods to approximate solutions. Future work could explore extending the framework to more complex systems, incorporating additional data modalities, scalability for high-dimensional systems, and expanding the text description scope.

Code for this work is available at `https://github.com/enegrini/MOL-LLM.git`.

Appendix A Dataset details

We list below the parametric equations and the corresponding parameters used to generated the data. For ordinary differential equations, we use SciPy's `solve_ivp` function to compute solutions. All ODEs solutions are generated for $t \in [0, 5]$. Partial differential equations and conservation laws are generated following the methodology described in [38], to which we refer for detailed information on specific solvers. All PDEs and conservation laws solutions are generated for $(t, x) \in [0, 5] \times [0, 2]$ except otherwise stated below. To create the dataset, we sample 100 parameter values in the range $[Q - 0.1Q, Q + 0.1Q]$ for each of the 52 parametric equations, resulting in a total of 5200 unique equations. For the ODE set we sample 50 initial conditions per equation, while for the PDE set we sample 100 initial conditions per equation. Additionally, for each of the equations, we generate 50 text descriptions using GPT-4. These descriptions focus on a variety of properties of the dynamics. In particular, some focus on scientific properties of the equation and its solution, such as order, linearity, main equation terms, or the presence of shocks or rarefactions in the solution. Others describe classical numerical methods which may be used to produce an approximate solution. Other focus on the natural dynamics that the equation describes, for example, the Lotka Volterra system describes the interactions between prey and predators in an ecosystem. Below is a complete list of equations with the corresponding parameters of interest.

ODE 1 (index 1):

$$\frac{du}{dt} = a \sin(2\pi t) u, \quad a = 1.$$

ODE 2 (index 2):

$$\frac{du}{dt} = ae^{-t} + b, \quad [a, b] = [1, 2].$$

ODE 3 (index 3):

$$\frac{du}{dt} = at^2 \cos(u) + bu, \quad [a, b] = [1, 3/10].$$

ODE 4 (index 4):

$$\frac{du}{dt} = a \sin\left(e^{-0.5t} \sin(3t)\right) + bu, \quad [a, b] = [2, 1/2].$$

ODE 5 (index 5):

$$\frac{du}{dt} = at \sin(u), \quad a = 3/2.$$

SIR system (index 6):

$$\begin{cases} \dfrac{dS}{dt} = -\beta SI, \\ \dfrac{dI}{dt} = \beta SI - \gamma I, \\ \dfrac{dR}{dt} = \gamma I, \end{cases} \qquad [\beta, \gamma] = [0.3, 0.1].$$

Neural dynamics system (index 7):

$$\begin{cases} \dfrac{dE}{dt} = \alpha E - \beta EI - \gamma E + 0.01 \sin(t), \\ \dfrac{dI}{dt} = \delta E - \epsilon I, \\ \dfrac{dH}{dt} = \theta I - \phi H, \end{cases} \qquad [\alpha, \beta, \gamma, \delta] = [0.2, 0.1, 0.05, 0.5].$$

Van der Pol oscillator (index 8):

$$\begin{cases} \dfrac{dx}{dt} = y, \\ \dfrac{dy}{dt} = \mu(1 - x^2)y - x, \end{cases} \qquad \mu = 2.$$

Lotka-Volterra system (index 9):

$$\begin{cases} \dfrac{dx}{dt} = \alpha x - \beta xy, \\ \dfrac{dy}{dt} = \delta xy - \gamma y, \end{cases} \qquad [\alpha, \beta, \gamma, \delta] = [1.5, 1, 3, 1].$$

FitzHugh-Nagumo system (index 10):

$$\begin{cases} \dfrac{dv}{dt} = v - \dfrac{v^3}{3} - w + I, \\ \dfrac{dw}{dt} = \dfrac{1}{\tau}\left(v + a - bw\right), \end{cases} \qquad [I, a, b, \tau] = [0, 0.7, 0.8, 0.8].$$

Brusselator system (index 11):

$$\begin{cases} \dfrac{dx}{dt} = A + x^2 y - (B+1)x, \\ \dfrac{dy}{dt} = Bx - x^2 y, \end{cases} \qquad [A, B] = [2, 4].$$

Duffing system (index 12):

$$\begin{cases} \dfrac{dx}{dt} = y, \\ \dfrac{dy}{dt} = -\delta y - \alpha x - \beta x^3, \end{cases} \qquad [\alpha, \beta, \delta] = [1, 0.2, 0.3].$$

Heat equation (index 13):

$$\frac{\partial u}{\partial t} = c_1 \frac{\partial^2 u}{\partial x^2}, \quad c_1 = 3 \times 10^{-3}.$$

Porous medium equation (index 14):

$$\frac{\partial u}{\partial t} = \frac{\partial^2}{\partial x^2}(u^m), \quad m = 2, 3, 4, \quad t_{final} = 0.1.$$

Klein-Gordon equation (index 15):

$$\frac{\partial^2 u}{\partial t^2} + q_2^2 q_1^4 u = q_1^2 \frac{\partial^2 u}{\partial x^2}, \quad [q_1, q_2] = [1, 0.1], \quad t_{final} = 1.$$

Sine-Gordon equation (index 16):

$$\frac{\partial^2 u}{\partial t^2} + q \sin(u) = \frac{\partial^2 u}{\partial x^2}, \quad q = 1, \quad t_{final} = 1.$$

Cahn-Hilliard equation (index 17):

$$\frac{\partial^2 u}{\partial t^2} + q^2 \frac{\partial^4 u}{\partial x^4} + 6\left(u \frac{\partial u}{\partial x}\right)_x = 0, \quad q = 0.01, \quad t_{final} = 0.5.$$

Korteweg-De Vries equation (index 18):

$$\frac{\partial u}{\partial t} + q^2 \frac{\partial^3 u}{\partial x^3} + u \frac{\partial u}{\partial x} = 0, \quad q = 0.022, \quad t_{final} = 1.$$

Advection equation (index 19):

$$\frac{\partial u}{\partial t} + q\frac{\partial u}{\partial x} = 0, \quad q = 0.5.$$

Wave equation (index 20):

$$\frac{\partial^2 u}{\partial t^2} = q\frac{\partial^2 u}{\partial x^2}, \quad q = 0.5, \quad t_{final} = 1.$$

Reaction-diffusion equation logistic (index 21):

$$\frac{\partial u}{\partial t} = q_1\frac{\partial^2 u}{\partial x^2} + q_2 R(u), \quad R(u) = u(1-u), \quad [q_1, q_2] = [3 \times 10^{-3}, 1].$$

Reaction-diffusion equation linear (index 22):

$$\frac{\partial u}{\partial t} = q_1\frac{\partial^2 u}{\partial x^2} + q_2 R(u), \quad R(u) = u, \quad [q_1, q_2] = [3 \times 10^{-3}, 0.1].$$

Reaction-diffusion equation bistable (index 23):

$$\frac{\partial u}{\partial t} = q_1\frac{\partial^2 u}{\partial x^2} + q_2 R(u), \quad R(u) = u^2(1-u), \quad [q_1, q_2] = [3 \times 10^{-3}, 1].$$

Reaction-diffusion equation square logistic (index 24):

$$\frac{\partial u}{\partial t} = q_1\frac{\partial^2 u}{\partial x^2} + q_2 R(u), \quad R(u) = u^2(1-u)^2, \quad [q_1, q_2] = [3 \times 10^{-3}, 1].$$

Burgers' equation (index 25):

$$\frac{\partial u}{\partial t} = -q_1\big(f(u)\big)_x + \frac{q_2}{\pi}\frac{\partial^2 u}{\partial x^2}, \quad f(u) = \frac{1}{2}u^2, \quad [q_1, q_2] = [1, 0.01].$$

Inviscid Burgers (index 26):

$$\frac{\partial u}{\partial t} = -k\big(f(u)\big)_x, \quad f(u) = \frac{1}{2}u^2, \quad k = 1.$$

Conservation law linear flux (index 27):

$$\frac{\partial u}{\partial t} = -q_1\big(f(u)\big)_x + \frac{q_2}{\pi}\frac{\partial^2 u}{\partial x^2}, \quad f(u) = u, \quad [q_1, q_2] = [1, 0.01].$$

Conservation law cubic flux (index 28):

$$\frac{\partial u}{\partial t} = -q_1\big(f(u)\big)_x + \frac{q_2}{\pi}\frac{\partial^2 u}{\partial x^2}, \quad f(u) = \frac{1}{3}u^3, \quad [q_1, q_2] = [1, 0.01].$$

Inviscid conservation law cubic flux (index 29):

$$\frac{\partial u}{\partial t} = -k\big(f(u)\big)_x, \quad f(u) = \frac{1}{3}u^3, \quad k = 1.$$

Conservation law sine flux (index 30):

$$\frac{\partial u}{\partial t} = -q_1\big(f(u)\big)_x + \frac{q_2}{\pi}\frac{\partial^2 u}{\partial x^2}, \quad f(u) = \sin(u), \quad [q_1, q_2] = [1, 0.01].$$

Inviscid conservation law sine flux (index 31):

$$\frac{\partial u}{\partial t} = -k\big(f(u)\big)_x, \quad f(u) = \sin(u), \quad k = 1.$$

Conservation law cosine flux (index 32):

$$\frac{\partial u}{\partial t} = -q_1\big(f(u)\big)_x + \frac{q_2}{\pi}\frac{\partial^2 u}{\partial x^2}, \quad f(u) = \cos(u), \quad [q_1, q_2] = [1, 0.01].$$

Inviscid conservation law cosine flux (index 33):

$$\frac{\partial u}{\partial t} = -k\big(f(u)\big)_x, \quad f(u) = \cos(u), \quad k = 1.$$

Fokker-Plank (index 34):

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} - \frac{D}{k_B T}\big(\nabla U(x) u\big)_x.$$

where $D = k_B T/\gamma, k_B \approx 1.380649 \times 10^{-23}$ is the Boltzmann constant, $T = 300$ is absolute temperature and $\gamma = 6\pi\eta r$ represents the drag coefficient, $r = 0.1 \times 10^{-6}$. The parameter of interest (randomized) is the fluid viscosity $\eta = 10^{-3}$. The potential is defined as

$$U(x) = 5 \times 10^{-21} \cos\left(\frac{x}{0.1 \times 10^{-6}}\right),$$

and the values are set to $t_{final} = 0.1, x_{final} = 2 \times 10^{-6}$.

Acknowledgments

The authors thank the reviewers for their helpful feedback.

E. Negrini, Y. Liu, and H. Schaeffer are supported in part by the AFOSR MURI FA9550-21-1-0084, by the NSF DMS 2427558, and by the NSF DMS 2331033. L. Yang is supported by the NUS Presidential Young Professorship. S. Osher is partially funded by the STROBE NSF STC887 DMR 1548924, by the NSF 00065369, and by the ONR N00014-20-1-2787.

References

[1] J. Achiam et al., GPT-4 technical report, *arXiv:2303.08774*, 2023.

[2] D. Bahdanau, K. Cho, and Y. Bengio, Neural machine translation by jointly learning to align and translate, *arXiv:1409.0473*, 2014.

[3] T. Brown et al., Language models are few-shot learners, in: *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 33:1877–1901, 2020.

[4] S. Cao, Choose a transformer: Fourier or Galerkin, in: *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 34:24924–24940, 2021.

[5] Y. Cao, Y. Liu, L. Yang, R. Yu, H. Schaeffer, and S. Osher, VICON: Vision in-context operator networks for multi-physics fluid dynamics prediction, *arXiv:2411.16063*, 2024.

[6] T. Chen and H. Chen, Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems, *IEEE Trans. Neural Netw.*, 6(4):911–917, 1995.

[7] W. Chen, J. Song, P. Ren, S. Subramanian, D. Morozov, and M. W. Mahoney, Data-efficient operator learning via unsupervised pretraining and in-context learning, *arXiv:2402.15734*, 2024.

[8] C. O. de Burgh-Day and T. Leeuwenburg, Machine learning for numerical weather and climate modelling: A review, *Geosci. Model Dev.*, 16(22):6433–6477, 2023.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Vol. 1, Association for Computational Linguistics, 4171–4186, 2019.

[10] A. Dosovitskiy, An image is worth 16x16 words: Transformers for image recognition at scale, *arXiv: 2010.11929*, 2020.

[11] D. Driess et al., PaLM-E: An embodied multimodal language model, *arXiv:2303.03378*, 2023.

[12] D. Feng, C. Haase-Schütz, L. Rosenbaum, H. Hertlein, C. Glaeser, F. Timm, W. Wiesbeck, and K. Dietmayer, Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges, *IEEE Trans. Intell. Transp. Syst.*, 22(3):1341–1360, 2020.

[13] R. Firoozi et al., Foundation models in robotics: Applications, challenges, and the future, *Int. J. Robot. Res.*, 44(5):701–739, 2025.

[14] S. Golkar et al., xVal: A continuous number encoding for large language models, *arXiv:2310.02989*, 2023.

[15] X. Han et al., Pre-trained models: Past, present and future, *AI Open*, 2:225–250, 2021.

[16] M. Herde, B. Raonić, T. Rohner, R. Käppeli, R. Molinaro, E. Bézenac, and S. Mishra, Poseidon: Efficient foundation models for PDEs, *arXiv:2405.19101*, 2024.

[17] A. Jadon, A. Patil, and S. Jadon, A comprehensive survey of regression-based loss functions for time series forecasting, in: *International Conference on Data Management, Analytics* & *Innovation*, Springer, 117–147, 2024.

[18] D. Jollie, J. Sun, Z. Zhang, and H. Schaeffer, Time-series forecasting, knowledge distillation, and refinement within a multimodal PDE foundation model, *arXiv:2409.11609*, 2024.

[19] I. Kulikov, A. H. Miller, K. Cho, and J. Weston, Importance of search and evaluation strategies in neural dialogue modeling, *arXiv:1811.00907*, 2018.

[20] L. H. Li, M. Yatskar, D. Yin, C.-J. Hsieh, and K.-W. Chang, VisualBERT: A simple and performant baseline for vision and language, *arXiv:1908.03557*, 2019.

[21] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, Fourier neural operator for parametric partial differential equations, *arXiv:2010.08895*, 2020.

[22] G. Lin, C. Moya, and Z. Zhang, Accelerated replica exchange stochastic gradient Langevin diffusion enhanced bayesian DeepONet for solving noisy parametric PDEs, *arXiv:2111.02484*, 2021.

[23] Y. Liu, J. Sun, X. He, G. Pinney, Z. Zhang, and H. Schaeffer, PROSE-FD: A multimodal PDE foundation model for learning multiple operators for forecasting fluid dynamics, *arXiv:2409.09811*, 2024.

[24] Y. Liu, J. Zhang, L. Fang, Q. Jiang, and B. Zhou, Multimodal motion prediction with stacked transformers, in: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, 7573–7582, 2021.

[25] Y. Liu, Z. Zhang, and H. Schaeffer, PROSE: Predicting multiple operators and symbolic expressions using multimodal transformers, *Neural Networks*, 180:106707, 2024.

[26] C. Lorsung, Z. Li, and A. B. Farimani, Physics informed token transformer for solving partial differential equations, *Mach. Learn.: Sci. Technol.*, 5(1):015032, 2024.

[27] J. Lu, D. Batra, D. Parikh, and S. Lee, VilBERT: Pretraining task-agnostic visiolinguistic representations

for vision-and-language tasks, in: *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 32:13–23, 2019.

[28] L. Lu, P. Jin, and G. E. Karniadakis, DeepOnet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators, *arXiv:1910.03193*, 2019.

[29] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, *Nat. Mach. Intell.*, 3(3):218–229, 2021.

[30] M. Ott, M. Auli, D. Grangier, and M. Ranzato, Analyzing uncertainty in neural machine translation, in: *International Conference on Machine Learning*, PMLR, 3956–3965, 2018.

[31] S. J. Pan and Q. Yang, A survey on transfer learning, *IEEE Trans. Knowl. Data Eng.*, 22(10):1345–1359, 2009.

[32] A. Poznyak, I. Chairez, and T. Poznyak, A survey on artificial neural networks application for identification and control in environmental engineering: Biological and chemical systems with uncertain models, *Annu. Rev. Control*, 48:250–272, 2019.

[33] A. Radford et al., Language models are unsupervised multitask learners, *OpenAI Blog*, 1(8):9, 2019.

[34] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, Zero-shot text-to-image generation, in: *International Conference on Machine Learning*, PMLR, 8821–8831, 2021.

[35] H. Schaeffer, Learning partial differential equations via data discovery and sparse optimization, *Proc. R. Soc. A: Math. Phys. Eng. Sci.*, 473(2197):20160446, 2017.

[36] O. B. Sezer, M. U. Gudelek, and A. M. Ozbayoglu, Financial time series forecasting with deep learning: A systematic literature review: 2005-2019, *Appl. Soft Comput.*, 90:106181, 2020.

[37] C. Sun, A. Myers, C. Vondrick, K. Murphy, and C. Schmid, VideoBERT: A joint model for video and language representation learning, in: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, IEEE, 7464–7473, 2019.

[38] J. Sun, Y. Liu, Z. Zhang, and H. Schaeffer, Towards a foundation model for partial differential equation: Multi-operator learning and extrapolation, *arXiv:2404.12355*, 2024.

[39] J. Sun, Z. Zhang, and H. Schaeffer, LEMON: Learning to learn multi-operator networks, *arXiv: 2408.16168*, 2024.

[40] H. Tan and M. Bansal, LXMERT: Learning cross-modality encoder representations from transformers, *arXiv:1908.07490*, 2019.

[41] M. Tan, M. A. Merrill, V. Gupta, T. Althoff, and T. Hartvigsen, Are language models actually useful for time series forecasting?, in: *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 37:60162–60191, 2024.

[42] H. Touvron et al., LLaMA: Open and efficient foundation language models, *arXiv:2302.13971*, 2023.

[43] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, Attention is all you need, in: *Proceedings of the 31st International Conference on Neural Information Processing Systems*, Curran Associates, Inc., 6000–6010, 2017.

[44] H. Wang et al., Recent advances on machine learning for computational fluid dynamics: A survey, *arXiv:2408.12171*, 2024.

[45] P. Xu, X. Zhu, and D. A. Clifton, Multimodal learning with transformers: A survey, *IEEE Trans. Pattern Anal. Mach. Intell.*, 45(10):12113–12132, 2023.

[46] L. Yang, S. Liu, T. Meng, and S. J. Osher, In-context operator learning with data prompts for differential equation problems, *Proc. Natl. Acad. Sci. USA*, 120(39):e2310142120, 2023.

[47] L. Yang, S. Liu, and S. J. Osher, Fine-tune language models as multi-modal differential equation solvers, *arXiv:2308.05061*, 2023.

[48] L. Yang and S. J. Osher, PDE generalization of in-context operator networks: A study on 1D scalar nonlinear conservation laws, *J. Comput. Phys.*, 519:113379, 2024.

[49] Z. Ye, X. Huang, L. Chen, Z. Liu, B. Wu, H. Liu, Z. Wang, and B. Dong, PDEformer-1: A foundation model for one-dimensional partial differential equations, *arXiv:2407.06664*, 2024.

[50] Q. Zhang et al., Scientific large language models: A survey on biological & chemical domains, *arXiv:2401.14656*, 2024.

[51] Z. Zhang, C. Moya, W. T. Leung, G. Lin, and H. Schaeffer, Bayesian deep operator learning for homogenized to fine-scale maps for multiscale PDE, *Multiscale Model. Simul.*, 22(3):956–972, 2024.