PDExplain - Contextual Modeling of PDEs in the Wild

Ori Linial
Technion, Israel Institute of Technology
Haifa, Israel
`linial04@gmail.com`

Orly Avner & Dotan Di Castro
Bosch Center for Artificial Intelligence,
Haifa, Israel

Abstract

We propose an explainable method for solving Partial Differential Equations by using a contextual scheme called *PDExplain*. During the training phase, our method is fed with data collected from an operator-defined family of PDEs accompanied by the general form of this family. In the inference phase, a minimal sample collected from a phenomenon is provided, where the sample is related to the PDE family but not necessarily to the set of specific PDEs seen in the training phase. We show how our algorithm can predict the PDE solution for future timesteps. Moreover, our method provides an explainable form of the PDE, a trait that can assist in modelling phenomena based on data in physical sciences. To verify our method, we conduct extensive experimentation, examining its quality both in terms of prediction error and explainability.

1 Introduction

Many scientific fields use the language of Partial Differential Equations (PDEs; Evans, 2010) to describe the physical laws governing observed natural phenomena with spatio-temporal dynamics. Typically, a PDE system is derived from first principles and a mechanistic understanding of the problem after experimentation and data collection by domain experts of the field. Solving a PDE model could provide users with crucial information on how a signal evolves over time and space, and could be used for both prediction and control tasks.

While solving PDEs holds great value, it might still be a difficult task in many cases. For many complex real-world phenomena, we might only know some of the dynamics of the system. For example, an expert might tell us that a heat equation PDE has a specific functional form but we do not know the values of the diffusion and drift coefficient functions. In this paper we focus mainly on this case.

There are different ways of solving PDEs when data is available. In Figure 1 we illustrate the spectrum of approaches to the problem of PDEs modeling and their solutions. The horizontal axis represents the amount of mechanistic knowledge required by each approach, i.e., how much prior knowledge we have on the source of the data in terms of the PDE structure. The approaches hovering above the axis are those that employ available data, while those below the axis only use mechanistic knowledge. We describe the different approaches in detail in Section 3.

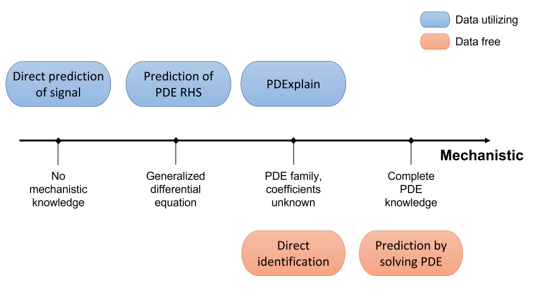


Figure 1: Mechanistic and data driven approaches to PDE modeling.

In recent years, with the rise of Deep Learning (DL; LeCun et al., 2015), novel methods for solving numerically-challenging PDEs were devised. In general, DL based approaches consume the observed data and learn a black-box model of the given problem that can then be used to provide predictions for the dynamics. While this set of solutions has been shown to perform successfully on many tasks, it still suffers from two crucial drawbacks: (1) It offers no explainability as to why the predictions were made, and (2) it usually performs very poorly when extrapolating to unseen data.

In this paper, we offer a new hybrid modelling Kurz et al. (2022) approach that can benefit from both worlds: it can use the vast amount of data collected on one hand, and utilize the partially known PDEs describing the natural phenomena observed on the other hand. In addition, it can learn several contexts, therefore, employing the generalization capabilities of DL models, enabling a zero-shot learning Palatucci et al. (2009). Specifically, our model is given a general functional form of the PDE (i.e., which derivatives are used and what the form of the coefficient functions is), consumes the observed data, and outputs the unknown coefficient functions. Then, we can then use off-the-shelf PDE solvers (e.g., PyPDE[1]) to solve and create predictions of the given task forward in time for any horizon.

Another key feature of our approach is that it consumes the spatio-temporal input signals required for training in an unsupervised manner, namely the coefficient functions that created the signals in the train set are unknown. This is achieved by combining an autoencoder (AE; Kramer, 1991) architecture with a loss defined using the functional form of the PDE. As a result of this feature, large amounts of training data for our algorithm can be easily acquired. Moreover, our ability to generalize to data corresponding to a PDE whose coefficients did not appear in the train set, enables the use of synthetic data for training.

We summarize our contribution as follows:

1. Harnessing the information contained in large datasets belonging to a phenomenon which is related to a PDE functional family in an unsupervised manner.
2. Proposing a DL encoding scheme for the context conveyed in such datasets, enabling generalization for prediction of unseen samples based on minimal input.
3. Extensive experimentation with the proposed scheme.

2 METHOD

We handle a dataset of spatio-temporal signals, generated by an underlying PDE. The coefficient functions determining the exact PDE are unknown and may be different for each collection of data. Our goal is to estimate these coefficient functions and provide reliable predictions of the future time steps of the observed phenomenon. The proposed method comprises three subsequent parts: (1) Creating a compact representation of the given signal, (2) estimating the PDE coefficients, and (3) solving the PDE using the acquired knowledge. Key features of the proposed method are lack of supervision, since the coefficient values of the PDEs represented in the train set are *unknown even at training time*, and explainability, thanks to the explicit prediction of the coefficient functions.

2.1 PROBLEM FORMULATION

Let $u(x,t)$ denote a signal with support $x \in [0,L]$ and $t \in [0,T]$. We refer to this as the *complete* signal. We define $u^c(x,t)$ to be a partial input signal, a *patch*, with support $x \in [0,L]$, $t \in [0,t_0]$, $0 < t_0 < T$. The superscript c stands for context.

We assume the signal $u(x,t)$ is the solution of a k-order PDE of the general form

$$\frac{\partial u}{\partial t} = \sum_{l=1}^{k} p_l(x,t,u)\frac{\partial u^l}{\partial x^l} + p_0(x,t,u), \tag{1}$$

where $p = (p_0, \ldots, p_k)$ is the vector of coefficient functions. We refer to a family of PDEs characterized by a vector p as an operator $F(p,u)$, where solving $F(p,u) = 0$ yields solutions of the PDE Wang & Yu (2021).

Our solution is a concatenation of two neural networks, which we call *PDExplain*. Its input is a patch $u^c(x,t)$ that solves a PDE of a *known* operator F with an *unknown* coefficient vector p, and its output is an estimated coefficient vector vector $\hat{p}$. We feed this vector into an off-the-shelf PDE solver together with the operator $F(p,u)$ to obtain the predicted signal $\hat{u}(x,t)$, $0 \le t \le T$.

[1]`https://pypde.readthedocs.io/en/latest/`

2.2 PDEXPLAIN INFERENCE AND TRAINING

Our inference process is presented in Fig. 2b. Its input is a patch $u^c(x,t)$ and an operator F. The patch is fed into PDExplain, generating the estimated coefficients $\hat{p}$.The PDE solver then uses this estimate to predict the complete signal, $\hat{u}(x,t)$.

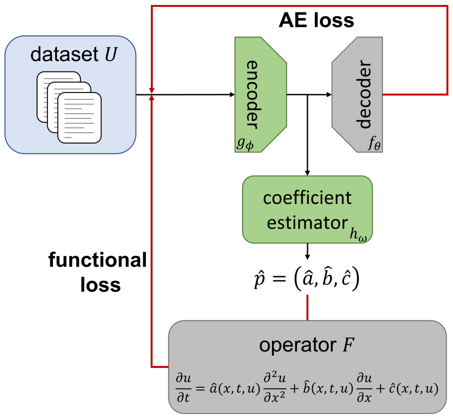
The patch $u^c(x,t)$ serves as an initial condition for the prediction and represents the dynamics of the signal for estimating the PDE coefficients. We refer to it as "context". The ratio of the context is denoted by ρ, such that $t_0 = \rho T$, and is a hyper-parameter of our algorithm, discussed in Section C.2.

The training process is presented in Fig. 2a. Its input is a dataset U that consists of N complete signals $\{u_i(x,t)\}_{i=1}^N$ which are solutions of N PDEs that share an operator F but have unique coefficient vectors $\{p_i\}_{i=1}^N$. The support of the signals is $x \in [0,L]$ and $t \in [0,T]$.

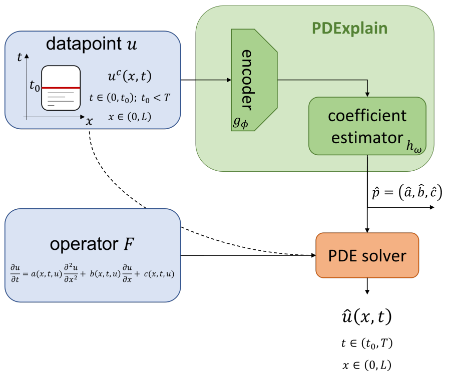
The PDExplain scheme is composed of two components: (1) an encoder and (2) a coefficient estimator. The encoder's goal is to capture the dynamics driving the signal u_i, creating a compact representation for the coefficient estimator. The encoder loss is the standard autoencoder reconstruction loss Hinton & Salakhutdinov (2006), namely the objective is $\min\limits_{\theta,\phi} \mathcal{L}_{\text{AE}} = \min\limits_{\theta,\phi} \sum_{i=1}^N \text{loss}(u_i^c - f_\theta(g_\phi(u_i^c)))$, where f_θ is the decoder, g_ϕ is the encoder and $\text{loss}(\cdot,\cdot)$ is a standard loss function.

The second component is the coefficient estimator. Together with the operator F, its output forms the functional objective: $\min\limits_{\omega} \mathcal{L}_{\text{coef}} = \min\limits_{\omega} \sum_{i=1}^N \|F(\hat{p}_\omega, u_i^c)\|^2$, where ω represents the parameters of the coefficient estimator network, and $\hat{p}$ is the estimator of p, acquired by applying the network h_ω to the output of the encoder.

The two components are trained simultaneously, and the total loss is a weighted sum of the autoencoder and coefficients losses: $\mathcal{L} = \alpha \cdot \mathcal{L}_{\text{AE}} + (1-\alpha) \cdot \mathcal{L}_{\text{coef}}$, where $\alpha \in (0,1)$ is a hyper-parameter. The complete algorithm is presented in Algorithm 2, in the appendix.


AE loss
dataset U
encoder
g_ϕ
decoder
f_θ
coefficient estimator
h_ω
functional loss
$\hat{p} = (\hat{a}, \hat{b}, \hat{c})$
operator F
$\frac{\partial u}{\partial t} = \hat{a}(x,t,u)\frac{\partial^2 u}{\partial x^2} + \hat{b}(x,t,u)\frac{\partial u}{\partial x} + \hat{c}(x,t,u)$

(a) Training process.


datapoint u
t
t_0
x
$u^c(x,t)$
$t \in (0,t_0);\ t_0 < T$
$x \in (0,L)$
PDExplain
encoder
g_ϕ
coefficient estimator
h_ω
$\hat{p} = (\hat{a}, \hat{b}, \hat{c})$
operator F
$\frac{\partial u}{\partial t} = a(x,t,u)\frac{\partial^2 u}{\partial x^2} + b(x,t,u)\frac{\partial u}{\partial x} + c(x,t,u)$
PDE solver
$\hat{u}(x,t)$
$t \in (t_0,T)$
$x \in (0,L)$

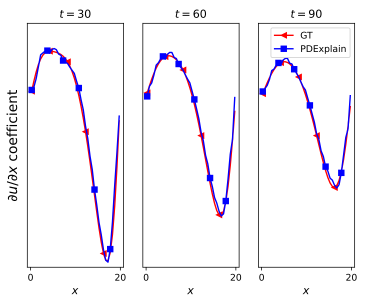
(b) Inference process. Dashed line is initial condition, taken from $u^c(x,t)$.

3 EXPERIMENTS

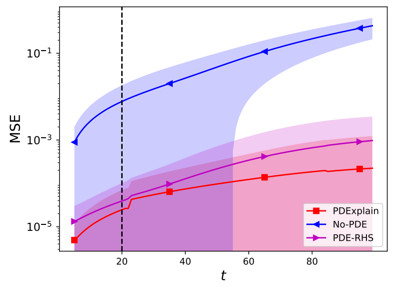
In this section we demonstrate the capabilities of PDExplain and compare it to several baselines on the Burgers' general equation of the form:

$$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2} + b(u)\frac{\partial u}{\partial x}. \tag{2}$$

This PDE describes a quasi-linear Burgers' equation with $b(x,t,u) = -u$, as in Bateman (1915). The dataset we use includes $10,000$ samples of size $100 \times 40[t\,\text{points} \times x\,\text{points}]$, where each sample is a signal generated from a PDE with different coefficients unknown to the algorithms a priori. We stress the fact that the test set contains signals generated by PDEs with coefficient vectors that *do not* appear in the training data, resulting in a zero-shot prediction problem.



(a) Estimation of the coefficient function $b(x, t, u)$. PDExplain manages to accurately estimate the spatio-temporal dynamics of the coefficient, based on a context ratio of $\rho = 0.2$.



(b) Prediction error vs. prediction horizon. No-PDE (blue) applies a purely data-driven prediction. PDE-RHS (magenta) predicts the right-hand-side of an equation which is then solved. Dashed vertical line is context ratio.

Figure 4: Burgers' PDE experiment as presented in Eq. 2 with $\rho = 0.20$

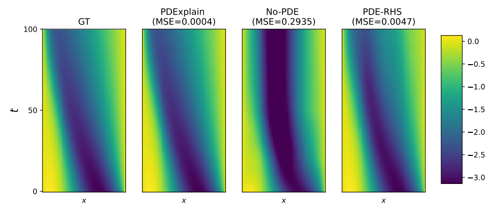


Figure 3: A solution of the Burgers' equation (prediction MSE appears in the title). The left panel displays the ground truth (GT), next to it the error-minimizing PDExplain prediction. No-PDE suffers from the largest error, which can be explained by its total lack of mechanistic knowledge. PDE-RHS achieves a relatively low MSE, but the quality of its prediction decreases over time.

In our experiments we implement three algorithms, in addition to our proposed PDExplain, corresponding to the different approaches presented in Figure 1. For the sake of fair comparison, all of the data utilizing approaches include a trainable encoder, similar to the one introduced in Section 2.2. Additional information about the algorithms, their implementation and dataset creation in the appendix.

Figure 4b demonstrates the clear advantage of our approach, which becomes increasingly larger as the prediction horizon increases (note the logarithmic scale of the vertical axis, representing the MSE of prediction). Since PDExplain harnesses both mechanistic knowledge and training data, it is able to predict the signal $\hat{u}(x, t)$ several timesteps ahead, while keeping the error to a minimum. We demonstrate a signal $u(x, t)$ from the test set and its prediction $\hat{u}(x, t)$ in Figure 3. As can be seen both visually and from the value of the MSE (in each panel's title), our approach yields a prediction that stays closest to the ground truth (GT), even as time advances and the prediction horizon increases. In Figure 4a we focus on the ability to accurately predict coefficient functions with spatio-temporal dynamics, specifically in this case - the coefficient $b(x, t, u)$ of Eq. 2. The different panels corresponds to different points in time, showing that the coefficient estimator tracks the temporal evolution successfully. In the appendix, we provide an additional experiment on a different family of PDEs that demonstrated similar results. We also provide additional analysis regarding how train set size and context ratio affect the results of the PDExplain algorithm.

4 Conclusion

In this work we introduce a new hybrid modelling approach, combining mechanistic knowledge with data. The knowledge we assume is in the form of a PDE family, without specific parameter values. The dataset we rely on is a collection of spatio-temporal signals belonging to the same PDE family, with different parameters. Unlike other schemes, we do not require knowledge of the parameters of the PDE generating our train data. We conduct extensive experiments, comparing our scheme to other solutions and testing its performance in different regimes. It achieves good results in the zero-shot learning problem, and is robust to different values of hyper-parameters.

REFERENCES

Christian Aarset, Martin Holler, and Tram Thi Ngoc Nguyen. Learning-informed parameter identification in nonlinear time-dependent pdes. *arXiv preprint arXiv:2202.10915*, 2022.

Harry Bateman. Some recent researches on the motion of fluids. *Monthly Weather Review*, 43(4): 163–170, 1915.

Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers. *arXiv preprint arXiv:2202.03376*, 2022.

Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

Kathleen Champion, Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116 (45):22445–22451, 2019.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Guozhi Dong, Michael Hintermüller, and Kostas Papafitsoros. Optimization with learning-informed differential equation constraints and its applications. *ESAIM: Control, Optimisation and Calculus of Variations*, 28:3, 2022.

Mengge Du, Yuntian Chen, and Dongxiao Zhang. Discover: Deep identification of symbolic open-form pdes via enhanced reinforcement-learning. *arXiv preprint arXiv:2210.02181*, 2022.

Lawrence C Evans. *Partial differential equations*, volume 19. American Mathematical Soc., 2010.

Hennes Hajduk, Dmitri Kuzmin, and Vadym Aizinger. *Bathymetry Reconstruction Using Inverse ShallowWater Models: Finite Element Discretization and Regularization*. Springer, 2020.

Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Matthieu Kirchmeyer, Yuan Yin, Jérémie Donà, Nicolas Baskiotis, Alain Rakotomamonjy, and Patrick Gallinari. Generalizing to new physical systems via context-informed dynamics model. *arXiv preprint arXiv:2202.01889*, 2022.

Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

Stefan Kurz, Herbert De Gersem, Armin Galetzka, Andreas Klaedtke, Melvin Liebsch, Dimitrios Loukrezis, Stephan Russenschuck, and Manuel Schmidt. Hybrid modeling: towards the next level of scientific computing in engineering. *Journal of Mathematics in Industry*, 12(1):1–12, 2022.

Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 2015.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *arXiv preprint arXiv:2111.03794*, 2021.

Ori Linial, Neta Ravid, Danny Eytan, and Uri Shalit. Generative ode modeling with known unknowns. In *Proceedings of the Conference on Health, Inference, and Learning*, pp. 79–94, 2021.

Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning pdes from data. In *International Conference on Machine Learning*, pp. 3208–3216. PMLR, 2018.

Zichao Long, Yiping Lu, and Bin Dong. Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 2019.

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

Aurel Neic, Fernando O Campos, Anton J Prassl, Steven A Niederer, Martin J Bishop, Edward J Vigmond, and Gernot Plank. Efficient computation of electrograms and ecgs in human whole heart simulations using a reaction-eikonal model. *Journal of computational physics*, 346:191–211, 2017.

Mark Palatucci, Dean Pomerleau, Geoffrey E Hinton, and Tom M Mitchell. Zero-shot learning with semantic output codes. *Advances in neural information processing systems*, 22, 2009.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

Samuel H Rudy, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Data-driven discovery of partial differential equations. *Science advances*, 3(4):e1602614, 2017.

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.

John C Strikwerda. *Finite difference schemes and partial differential equations*. SIAM, 2004.

Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Dan MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. Pdebench: An extensive benchmark for scientific machine learning. *arXiv preprint arXiv:2210.07182*, 2022.

Rui Wang and Rose Yu. Physics-guided deep learning for dynamical systems: A survey. *arXiv preprint arXiv:2107.01272*, 2021.

Tsung-Yen Yang, Justinian P Rosca, Karthik R Narasimhan, and Peter Ramadge. Learning physics constrained dynamics using autoencoders. In *Advances in Neural Information Processing Systems*, 2022.

Kirill Zubov, Zoe McCarthy, Yingbo Ma, Francesco Calisto, Valerio Pagliarino, Simone Azeglio, Luca Bottero, Emmanuel Luján, Valentin Sulzer, Ashutosh Bharambe, et al. Neuralpde: Automating physics-informed neural networks (pinns) with error approximations. *arXiv preprint arXiv:2107.09443*, 2021.

A Related Work

Creating a neural-network based model for approximating the solution of a PDE has been studied in many related works over the years, and dates back more than a decade Lagaris et al. (1998). We divide deep learning based existing work by their ability to incorporate mechanistic knowledge in their models, and by the type of information that can be extracted from using them. An additional distinction between different approaches is their ability to handle datasets originating from different contexts. From a PDE perspective, a different context could refer to having data signals generated with different coefficients functions (p_l in Eq. equation 1). In many real-world applications, obtaining observed datasets originating from a single context is highly infeasible. For example, in cardiac electrophysiology Neic et al. (2017), patients differ in cardiac parameters like resistance and capacitance, thus representing different contexts. In fluid dynamics, the topography of the underwater terrain (bathymetry) differs from one sample to another Hajduk et al. (2020). A benchmark and dataset work (PDEBench) that provides a large amount of datasets governed by known PDEs has been released recently Takamoto et al. (2022), but each of the datasets provided by it is generated from a single constant function (i.e., all data samples have the same context).

The first line of work is purely data-driven based methods. These models come in handy especially when we observe a spatio-temporal phenomenon, but either don't have enough knowledge of the underlying PDE dynamics that generated the observed signal, or the known equations are too complicated to solve numerically (as explained thoroughly by Wang & Yu, 2021). Recent advances demonstrate successful prediction results that are both fast to compute (compared to numerically solving a PDE), and also shown to provide decent predictions even for PDEs with very high dimensions Brandstetter et al. (2022); Li et al. (2020); Han et al. (2018); Lu et al. (2019). However, the downside of this approach is not being able to infer the PDE coefficients, which may hold valuable information and explanations as to why the model formed its predictions.

The second type of data-driven methods are approaches that can utilize PDE forms known beforehand to some extent. Works that adopt this approach can usually utilize the given mechanistic knowledge and provide reliable predictions, ability to generalize to unseen data, and even, in some cases, reveal part of the underlying PDE coefficient functions. However, their main limitation is that they assume the entire training dataset is generated by a single coefficient function and only differ in the initial conditions (or possibly boundary conditions). PDE-NET Long et al. (2018), its followup PDE-NET2 Long et al. (2019), DISCOVER Du et al. (2022), PINO Li et al. (2021) and sparse-optimization methods Schaeffer (2017); Rudy et al. (2017) (expanding the idea originally presented on ODEs in Brunton et al. (2016); Champion et al. (2019)), are not given the PDE system, but instead aim to learn some representation of the underlying PDE as a linear combination of base functions and derivatives of the PDE state. PINN Raissi et al. (2019) and NeuralPDE Zubov et al. (2021) assume full knowledge of the underlying PDE including the its coefficients, and aim to replace the numerical PDE solver by a fast and reliable model. They also provide a scheme for finding the PDE parameters as scalars, but assume the entire dataset is generated by a single coefficient value, while we assume each sample is generated with different coefficient values and could be functions of time, space and state (as described in Eq. equation 1). Similarly, Learning-informed PDEs Dong et al. (2022); Aarset et al. (2022) suggest a method that assumes full knowledge of the PDE derivatives and their coefficient functions, and infers the free coefficient function (namely $p_0(x, t, u)$ in Eq. equation 1).

The last line of work, and closer in spirit to ours, includes context-aware methods that assume some mechanistic knowledge, with each sample in the train set generated by different PDE coefficients (we also refer to this concept as having different context) and initial conditions. CoDA Kirchmeyer et al. (2022) provides an ability to form predictions of signals with unseen contexts, but does not directly identify the PDE parameters. GOKU Linial et al. (2021) and ALPS Yang et al. (2022) provide context-aware inference of signals with ODE dynamics, when the observed signals are not the ODE variables directly.

B Implementation

We provide further information regarding the experiments described in Section 3. We ran all of the experiments on a single standard GPU, and all training algorithms took < 10 minutes to train.

Table 1: Characteristics of the implemented algorithms.

Approach	Mechanistic knowledge	Training data	Explainability
No PDE	-	+	-
PDE RHS	-	+	-
DI	+	-	+
PDExplain	+	+	+

Algorithm 1 PDExplain inference scheme

```
**Input:** patch $u^c(x,t)$, operator $F$, trained networks: decoder $g_\phi$, coefficient estimator $h_\omega$
$\hat{p} \leftarrow h_\omega(g_\phi(u^c))$
$\hat{u} \leftarrow \text{PDE\_solve}(F, \hat{p}, u^c(x, t = t_0))$
return $\hat{u}, \hat{p}$
```

Full code implementation for creating the datasets and implementing PDExplain and its baselines is available on github.com/orilinial/PDExplain.

B.1 Dataset creation

To create the dataset, we generated signals using the PyPDE package. Each signal was generated with different initial conditions sampled from a Gaussian process posterior, and a-priori known Dirichlet boundary conditions $u(x = 0) = u(x = L) = 0$. As discussed in Section A, we made an important change compared to other known methods: the PDE coefficients are uniformly sampled for each signal, instead of being constant, making the task much harder. In the constant coefficients experiment we generated the parameters from $a \sim U[0, 2]$, $b, c \sim U[-1, 1]$. For the Burgers' equation dataset we used $a \sim U[1, 2]$, and set $b(u) = -u$, both unknown to the algorithm a priori.

B.2 Algorithmic details

We present the inference scheme for PDExplain in Algorithm 1, and the full training algorithm in Algorithm 2.

In addition, we summarize the different baseline approaches in Table 1, adding the feature of explainability, available in schemes that explicitly estimate PDE coefficients. The algorithms are:

- "No PDE": Direct prediction of the signal $\hat{u}(x,t)$, given the partial signal (patch) $u^c(x,t)$ and an encoding of it, learned from training data. No use of mechanistic knowledge.
- "DI": Direct identification of the PDE coefficients from the partial signal $u^c(x,t)$ using a maximum likelihood approach, followed by solving the resulting PDE to predict the signal. Assumes PDE family is known, parameters unknown. No use of training data.
- "PDE-RHS": Prediction of the underlying PDE's right-hand-side based on the partial signal (patch) $u^c(x,t)$ and an encoding of it, followed by solving the resulting PDE to predict the signal. This approach that combines training data with minimal mechanistic knowledge, similar to that suggested by Chen et al. (2018).

B.3 Implementation details

All algorithms described in this paper except for DI (i.e., PDExplain, No-PDE and PDE-RHS) share the same context-extraction architecture. The architecture consists of an encoder-decoder network, both implemented as MLPs with 6 layers and 256 neurons in each layer. We experimented with removing the decoder and training the networks without the auto-encoder loss, but results proved to be poor. We found that concatenating the latent vector in the output of the encoder to the initial conditions of the signal $u(t = 0)$ greatly improved results and convergence time, since it encourages the encoder to focus on the dynamics of the observed signal, rather than the initial conditions of it.

Algorithm 2 Algorithm for training PDExplain

```
Input: dataset $U$, operator $F$, context ratio $\rho$, loss weight $\alpha$, number of epochs $N_e$
Init: random weights in encoder $g_\phi$, decoder $f_\theta$, coefficient estimator $h_\omega$
for epoch in $N_e$ do
    $\mathcal{L} \leftarrow 0$
    $U_N^c \leftarrow N$ random patches, one from each $u_i \in U$
    for $u_i^c$ in $U_N^c$ do
        $\hat{p}_i \leftarrow h_\omega(g_\phi(u_i^c))$
        $\mathcal{L} \leftarrow \mathcal{L} + \alpha \cdot (u_i^c - f_\theta(g_\phi(u_i^c)))^2 + (1-\alpha) \cdot \|F(\hat{p}_i, u_i^c)\|^2$
    end for
    $\phi, \theta, \omega \leftarrow \arg\min \mathcal{L}$
end for
```

The second part of each algorithm uses the latent vector as an input to a Context-To-Dynamics network.

In PDExplain we implemented this network as an MLP with 5 hidden layers, each with 1024 neurons. The output of the network is then the task-specific parameters with the correct shape. For the Burgers' equation for example, the output of the network is the parameter a, and a function $b(u)$ with the same shape as u.

In the No-PDE algorithm, the Context-To-Dynamics network consumes the current PDE state and the latent vector, and outputs the PDE state in the next time step. The optimization function for this algorithm therefore tries to minimize the prediction error of u_{t+1} in addition to the autoencoder loss.

The PDE-RHS algorithm is similar to the PDExplain algorithm as it tries to learn a derivative and not the predicted state. The Context-To-Dynamics network in this case consumes the latent vector and the current state, and outputs the right-hand-side of the PDE equation.

Both PDE-RHS and No-PDE baselines have the same architecture of the Context-To-Dynamics net as PDExplain, and only differ in the shape of the output. Sharing the same architecture allow us to carefully compare these methods and answer the question of how mechanistic knowledge can be used.

B.4 Numerical scheme

The partial derivatives in our algorithms are estimated using standard numerical schemes for each point in the patch. We choose discretization parameters Δx for the spatial axis and Δt for the temporal axis where we solve the PDE numerically on the grid points $\{(i\Delta x, j\Delta t)\}_{i=0,j=0}^{N_x,N_t}$ with $L = N_x \Delta x$ and $T = N_t \Delta t$. Let us denote the numerical solution with $\hat{u}_{i,j}$. We use the *forward-time central-space* scheme, so a second order scheme from equation 1 would be

$$\begin{aligned}\frac{\hat{u}_{i,j+1} - \hat{u}_{i,j}}{\Delta t} =& p_2(i,j,u(i,j)) \frac{\hat{u}_{i+1,j} - 2\hat{u}_{i,j} + \hat{u}_{i-1,j}}{\Delta x^2} \\ &+ p_1(i,j,u(i,j)) \frac{\hat{u}_{i+1,j} - \hat{u}_{i-1,j}}{2\Delta x} \\ &+ p_0(i,j,u(i,j))\end{aligned} \tag{3}$$

For ease of exposition we omit some details and refer the reader to Strikwerda (2004) for a complete explanation.

C Experiments

In this section we provide additional experiments demonstrating PDExplain capabilities on another family of PDEs:

$$\frac{\partial u}{\partial t} = a \frac{\partial^2 u}{\partial x^2} + b \frac{\partial u}{\partial x} + c, \tag{4}$$

which is a second Order PDE with constant coefficients (i.e., $p = (a, b, c)$ are constants).

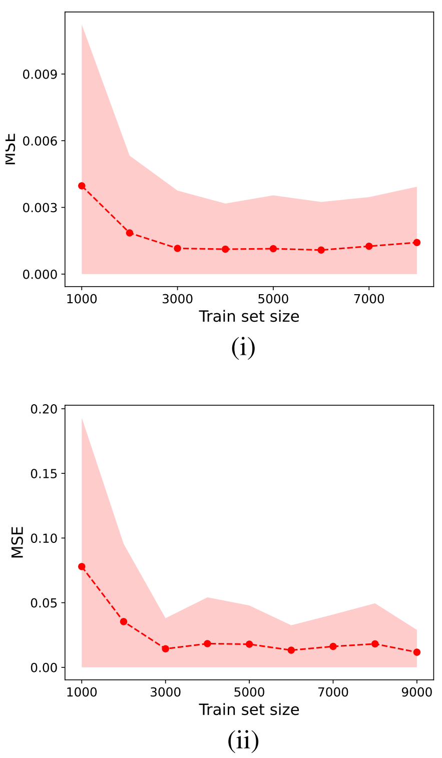
We start by analyzing the two parameters that characterize the PDExplain algorithm: train set size and context ratio, and continue by demonstrating the capabilities of PDExplain on another family of PDEs described in Eq. 4. Table 2 summarize the signal prediction and parameter estimation results for both Burgers' PDE, and the constant coefficients PDE.

C.1 Train set size

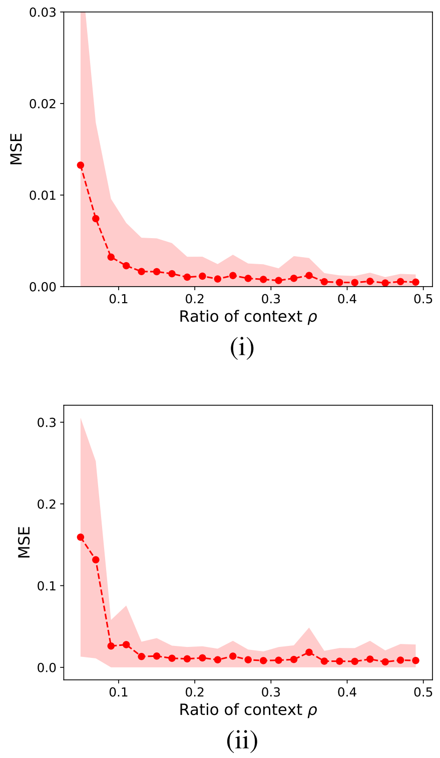
The train set size corresponds to N, the number of samples in dataset U of Algorithm 1. Figure 5a presents the decrease in the prediction and parameters error as we increase the train set size. This attests to the generalization achieved by the PDExplain architecture: as the train set grows and includes more samples with different values of coefficients, the ability to accurately estimate a new sample's parameters and predict its rollout improves. In this set of experiments, $3,000$ samples are generally enough to achieve a minimal error rate.

C.2 Context Ratio

Another hyper-parameter of our system is the context ratio. Figure 5b presents the results of an experiment in which we vary its value as defined in Section 2.2. Simply put, as the context size increases, PDExplain encodes more information regarding the input signal's dynamics, thus the improvement in signal and parameter value prediction. The error decreases rather quickly, and a context ratio of $0.15 - 0.2$ suffices for reaching a very low error, as is evident from the plots.



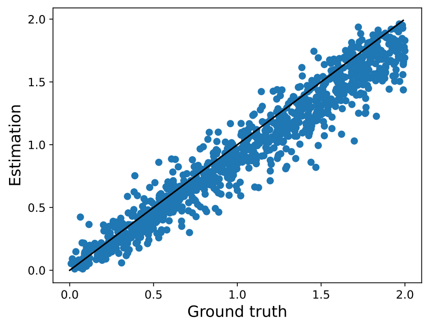
(a) Constant coefficients PDE: (i) Prediction error of signal vs. train set size and (ii) estimation error of parameter values vs. train set size. The error is calculated on a test set of 1000 samples.



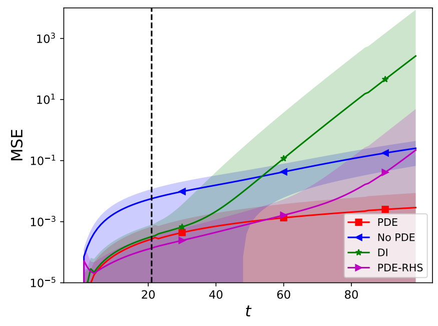
(b) Constant coefficients PDE: (i) Prediction error vs. ρ and (ii) estimation error of parameter values error vs. ρ. The error is calculated on a test set of 1000 samples.

C.3 Signal prediction and parameter estimation

Figure 6b displays a comparison between the different approaches to our problem. As before, the vertical axis of the plot is logarithmic, and the advantage of PDExplain over other approaches increases with the prediction horizon. In Figure 6a, we plot the estimated value of parameter a of Eq. 4, against its true value. The plot and the high value of R^2 demonstrate the low variance of our prediction, with a strong concentration of values along the $y = x$ line.



(a) Constant coefficients PDE: estimated value of the $\partial^2 u/\partial^2 x$ coefficient vs. ground truth, for entire test set ($R^2 = 0.93$).



(b) Prediction error vs. prediction horizon, different approaches. PDExplain (red) is our approach. No PDE (blue) learns an encoding for training data and applies a purely data-driven prediction. DI (green) is direct identification of PDE coefficients from datapoint, followed by solving the PDE (no training). PDE-RHS (magenta) learns an encoding for the training data and predicts the right-hand-side of an equation which is then solved. Dashed vertical line is the context ratio.

Figure 6: PDE with constant coefficients. $\rho = 0.21$.

Table 2: Results summary for both the coefficient identification and signals prediction task, on two experiments: constant coefficients equation and Burgers' equation. In both experiments we used $\rho = 0.2$, and report the MSE error. DI baseline in the constant coefficient experiment completely failed in predicting one of the test signals, so that measurement was omitted from the calculation to have a fair comparison.

Method	Parameters MSE	Prediction MSE
Constant PDE coefficients		
PDExplain	0.0116 ± 0.014	0.0014 ± 0.003
DI	0.0097 ± 0.022	$0.0067^{*} \pm 0.161$
No-PDE	N/A	0.0720 ± 0.051
PDE-RHS	N/A	0.0174 ± 0.326
Burgers' PDE		
PDExplain	0.0147 ± 0.0188	0.0001 ± 0.0004
DI	N/A	N/A
No-PDE	N/A	0.1303 ± 0.0839
PDE-RHS	N/A	0.0004 ± 0.0010