Diff-SINDy: A Differentiable framework for Discovering Partial Differential Equations from Data

Eddie Aljamal
ealjamal@umich.edu
University of Michigan
Ann Arbor, MI, USA

Thomas M. Baer
tbaer@umich.edu
University of Michigan
Ann Arbor, MI, USA

Devon Loomis
dloom@umich.edu
University of Michigan
Ann Arbor, MI, USA

ABSTRACT

Discovering the governing equations of complex nonlinear partial differential equations (PDEs) from data remains a fundamental challenge across scientific domains. Traditional approaches such as SINDy rely on fixed libraries of candidate functions, limiting flexibility, while Evo-SINDy relaxes this constraint but remains non-differentiable and depends on discrete derivative estimates that are sensitive to noise. We introduce **Diff-SINDy**, a fully differentiable framework for data-driven PDE discovery. Diff-SINDy employs a neural network to approximate the solution and uses automatic differentiation to compute spatial and temporal derivatives with improved accuracy and noise robustness. Physics-informed losses are used to train neural networks that model derivative coefficients as functions of space, time, and the solution itself. These learned coefficients are then distilled into interpretable closed-form expressions through symbolic regression.

1 INTRODUCTION

Since the advent of calculus, scientists have manually constructed partial differential equations (PDEs) to model physical systems by combining observed data with prior knowledge. This process is inherently complex: the search space of possible mathematical forms is NP-hard and remains vast even when substantial prior information is available. Reducing reliance on such priors only enlarges this space further, often resulting in combinatorial explosion. Automating PDE discovery with minimal assumptions would therefore simplify and accelerate the identification of governing equations and physical laws, making these tools accessible to non-experts.

Over the past two decades, Physics-Informed Neural Networks (PINNs) [16, 17] have significantly advanced the numerical modeling of known ODEs and PDEs across many scientific domains. Compared to classical numerical methods such as the finite element method (FEM), PINNs offer both computational efficiency and low memory usage. A PINN approximates the solution using a neural network and optimizes the composite loss

$$\mathcal{L} = \mathcal{L}_{\text{data}} + \mathcal{L}_{\text{PDE}} + \mathcal{L}_{\text{IC}} + \mathcal{L}_{\text{BC}}, \tag{1}$$

where $\mathcal{L}_{\text{data}}$ enforces agreement with observations, $\mathcal{L}_{\text{IC}}$ and $\mathcal{L}_{\text{BC}}$ impose initial and boundary conditions, and $\mathcal{L}_{\text{PDE}}$ ensures that derivatives computed via automatic differentiation (AD) satisfy the known PDE. In this work, we consider the reverse setting: given observations of an *unknown* PDE's solution, our goal is to discover its governing functional form by leveraging both data and PDE losses.

Data-driven equation discovery has long been a central challenge. Symbolic regression (SR) [2, 3, 8, 19, 20] with genetic programming (GP) aims to infer mathematical expressions $y = f(\mathbf{x})$ from data pairs $(\mathbf{x}, y)$ by composing unary and binary operators. Seminal works by Bongard, Lipson, and Schmidt [3, 19] demonstrated GP-based discovery of nonlinear ordinary differential equations and conservation laws from observed trajectories. However, evolutionary search methods are non-differentiable, scale poorly in high-dimensional spaces, and are susceptible to overfitting. The Sparse Identification of Nonlinear Dynamical Systems (SINDy) framework [4] mitigated these issues by restricting the candidate space to a predefined library of functions and using FEM-based derivative estimates with sparse regression. Evo-SINDy [11] further relaxed this constraint by incorporating derivative operators directly into the SR search space. Nevertheless, Evo-SINDy remains non-differentiable and depends on pre-computed FEM derivatives, making it sensitive to noise.

We introduce **Diff-SINDy**, a fully differentiable framework for discovering governing PDEs directly from data. Diff-SINDy uses a neural network to approximate the solution and employs AD to compute spatial and temporal derivatives accurately and robustly. An MLP architecture—as either a multi-task MLP, several separate MLPs, or a single unified MLP—predicts coefficients for a set of plausible derivative terms. In this study, we use the single MLP and the multi-task learning formulation: an MLP backbone feeding multiple shallow heads, each corresponding to one derivative (including the constant term). Training proceeds in two alternating stages. In the first stage, the solution MLP is optimized using the total loss (IC, BC, data, and PDE losses) while the coefficient network is frozen. In the second stage, the solution network is frozen and the coefficient model is trained solely using the sparsity-promoting PDE loss. After these alternating phases, inference is performed separately using PySR [8], which receives triplets of time, position, and neural-network-predicted solution values and returns closed-form expressions for the learned coefficients as functions of these inputs. This differentiable, function-library-free approach enables flexible and noise-resilient discovery of governing equations directly from data.

2 PROBLEM FORMULATION

Many dynamical systems and physical processes are governed by nonlinear partial differential equations (PDEs) with diverse structures and derivative orders. Although our framework extends naturally to higher-dimensional settings, we focus here on PDEs that are linear in their derivatives (i.e., no nonlinear operations on derivatives such as u_x^2), defined over one spatial and one temporal dimension, following the formulations in Chen et al. [7] and Jiang and Sun [11].

We first consider PDEs of the general form

$$u_t = f_0(x,t,u) + f_1(x,t,u)\,u_x + f_2(x,t,u)\,u_{xx} + f_3(x,t,u)\,u_{tt} + \cdots + f_j(x,t,u)\,u_{xxx} + \ldots, \quad (2)$$

where any derivative can appear on the left-hand side (LHS), provided that the same derivative does not also appear on the right-hand side (RHS).

A set of s candidate derivatives (augmented by ones to accommodate a forcing function)

$$\{1, u_x,\ u_{xx},\ u_{tt},\ \ldots,\ u_{xxx},\ \ldots\}$$

is pre-specified to define the library of possible terms. The associated coefficient functions

$$\{f_0,\ f_1,\ f_2,\ f_3,\ \ldots,\ f_j,\ \ldots,\ f_s\}$$

are unknown and may depend on (x,t) and the solution $u(x,t)$. This derivative library is far less restrictive than the function libraries required in SINDy [4] and PINN-SR [7], since most PDEs contain only a small number of derivative terms of modest order.

Analogously, PDEs with other dominant derivatives can be written as

$$u_{tt} = g_0(x,t,u) + g_1(x,t,u)\,u_x + g_2(x,t,u)\,u_t + g_3(x,t,u)\,u_{xx} + \cdots + g_j(x,t,u)\,u_{xxx} + \ldots, \quad (3)$$

with the derivative library and coefficient functions g_i defined in an analogous manner.

Given spatio-temporal samples (x,t), partially observed solution data $u(x,t)$, and a predefined library of candidate derivatives, the goal of Diff-SINDy is to recover a parsimonious closed-form PDE that explains the data. Specifically, Diff-SINDy learns symbolic expressions for the coefficient functions $f_i(x,t,u)$ or $g_i(x,t,u)$ for $i \in \{0,\ldots,s\}$, providing an interpretable and compact representation of the underlying governing equation.

3 RELATED WORKS

Diff-SINDy builds on a broad line of research in data-driven discovery of differential equations. Here we review the most relevant approaches and highlight their key characteristics.

The SINDy framework [4] identifies governing equations by constructing a library of candidate nonlinear functions and applying sparse regression to select the active terms. Its central innovation is the use of sparsity-promoting regularization, which yields parsimonious models that capture only the essential dynamics. SINDy also employs a sequential procedure to mitigate degeneracies: higher-order polynomial terms are introduced incrementally, allowing the correct structure to emerge before spurious interactions from higher-order modes can interfere. Despite its effectiveness, SINDy—along with other direct equation-discovery methods—remains sensitive to noise, particularly in data-limited regimes. Subsequent improvements using bootstrapping [10] and automatic differentiation [13] have enhanced robustness, but the approach remains fundamentally constrained by the fixed, incomplete function library, which limits performance as equation complexity increases.

Evo-SINDy [11] extends SINDy by relaxing the library restrictions through an evolutionary search over candidate functions with the addition of a derivative operator (using FEM). Its coevolutionary mechanism operates jointly with sparse regression, yielding more expressive models than standard evolutionary methods. However, like all genetic-programming–based techniques, Evo-SINDy is non-differentiable and thus more susceptible to noisy measurements.

Table 1: Comparison of Diff-SINDy's features and architecture with relevant related works

Feature	SINDy	PINN-SR	Evo-SINDy	**Diff-SINDy**
No Functional Library	✗	✗	✓	✓
Differentiability	✗	✓	✗	✓
AD Derivatives	✗	✓	✗	✓
Sparsity	✓	✓	✓	✓
No Pre-training	✓	✗	✗	✓

Earlier evolutionary approaches to PDE discovery [22] explored larger hypothesis spaces, but often suffered from high computational cost and two-stage workflows that decoupled structure discovery from coefficient estimation. Other methods [14] unified these steps but required manual tuning of regularization hyperparameters and multiple training runs for stability, leaving them sensitive to noise. In contrast, Diff-SINDy uses differentiable optimization during training and employs genetic programming only at inference.

Additional equation-discovery methods based on genetic programming [1, 19] focused primarily on discovering invariants or conservation laws rather than full dynamical models, making them architecturally and conceptually distinct from SINDy-type methods. More recently, variational formulations have been proposed for direct discovery, first for ODEs [15] and subsequently for PDEs [12]. These methods avoid explicit derivative estimation and thus offer improved noise robustness, but they still require a predefined dictionary of candidate derivatives and incur substantially higher computational cost than sparse-regression or evolutionary approaches.

Diff-SINDy is a fundamentally different framework for PDE discovery, as summarized in Table 1. It replaces the need for a large functional library with a compact derivative library, remains fully differentiable during gradient-based training through the use of automatic differentiation, and incorporates sparsity to promote parsimonious PDE expressions. In addition, it is computationally efficient and requires no pretraining.

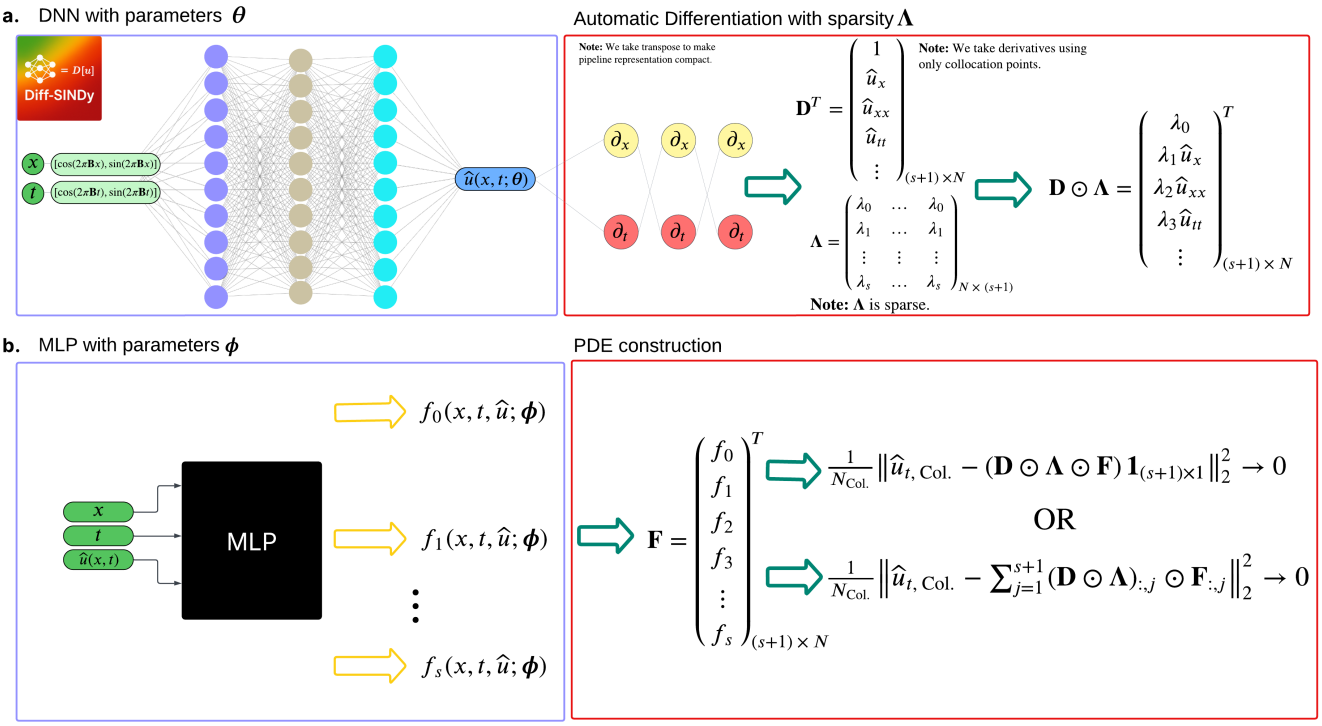
4 METHODS

We first describe the training procedure that produces pointwise estimates of the solution and coefficient functions in Section 4.1, followed by the inference stage in Section 4.2, where symbolic expressions are obtained using the genetic programming–based symbolic regression framework, PySR [8].

4.1 Training

Our training pipeline, illustrated in Figure 1, follows a PINN-inspired architecture where the solution is approximated by a neural network that enables AD. This leverages advances in neural network optimization and differentiable programming.

Given spatio-temporal coordinates (x,t), we first embed them using random Gaussian Fourier features [16, 17] with an embedding



c. Losses and Training

$$\mathcal{L}_{\text{IC}}(\theta; \overbrace{u_{\text{IC}}}^{\text{measured}}) = \frac{1}{N_{\text{IC}}} \| \overbrace{\hat{u}_{\text{IC}}}^{\text{predicted}} - u_{\text{IC}} \|_2^2 \qquad \mathcal{L}_{\text{Data}}(\theta; u_{\text{Data}}) = \frac{1}{N_{\text{Data}}} \| \hat{u}_{\text{Data}} - u_{\text{Data}} \|_2^2 \qquad \mathcal{L}_{\text{BC}}(\theta; u_{\text{BC}}) = \frac{1}{N_{\text{BC}}} \| \hat{u}_{\text{BC}} - u_{\text{BC}} \|_2^2$$

$$\mathcal{L}_{\text{PDE}}(\phi; x_{\text{Col.}}, t_{\text{Col.}}, u_{\text{Col.}}, \mathbf{\Lambda}) = \frac{1}{N_{\text{Col.}}} \left\| \hat{u}_{t,\text{Col.}} - (\mathbf{D} \odot \mathbf{\Lambda} \odot \mathbf{F})\, \mathbf{1}_{(s+1)\times 1} \right\|_2^2$$

Stage 1: Train solution with frozen sparsity and frozen coefficients model

$$\mathcal{L}(\theta; x, t, u, \overbrace{\phi}^{\text{unfrozen}}, \overbrace{\mathbf{\Lambda}}^{\text{frozen}}) = \lambda_{\text{IC}} \mathcal{L}_{\text{IC}} + \lambda_{\text{BC}} \mathcal{L}_{\text{BC}} + \lambda_{\text{Data}} \mathcal{L}_{\text{Data}} + \lambda_{\text{PDE}}(\text{epoch}) \mathcal{L}_{\text{PDE}}$$

Stage 2: Train coefficient model and sparsity with **STRidge** using collocation points

$$\mathcal{L}(\overbrace{\phi, \mathbf{\Lambda}}^{\text{unfrozen}}; x_{\text{Col.}}, t_{\text{Col.}}, u_{\text{Col.}}, \overbrace{\theta}^{\text{frozen}}) = \mathcal{L}_{\text{PDE}} + \beta \|\mathbf{\Lambda}\|_0$$

Figure 1: Diff-SINDy pipeline. **(a) Solution approximation (violet box):** A deep neural network (DNN) with parameters θ approximates the observed solution u from random Gaussian sinusoidal features using the embedding matrix $\mathbf{B}$ applied to the input pairs (x, t). **(Red box):** Automatic differentiation is used to compute a set of s candidate derivatives from the collocation points in the derivative library $\mathbf{D}$, retaining only the active terms via the sparsity matrix $\mathbf{\Lambda}$, parameterized by $\lambda_0, \dots, \lambda_s$. **(b) Coefficient estimation (violet box):** An MLP with parameters ϕ predicts the $s+1$ coefficients $f_0, \dots, f_s$ of the PDE (including a constant term), which are concatenated into the coefficient library $\mathbf{F}$. **(Red box):** These estimated coefficients are combined with the derivative library $\mathbf{D}$ and the sparse matrix $\mathbf{\Lambda}$ to minimize the PDE loss defined in Equation (13).

matrix $\mathbf{B}$:

$$\xi(x) = \left[\sin(2\pi \mathbf{B}x), \cos(2\pi \mathbf{B}x) \right], \tag{4}$$

$$\psi(t) = \left[\sin(2\pi \mathbf{B}t), \cos(2\pi \mathbf{B}t) \right]. \tag{5}$$

The observed solution $u(x, t)$ is approximated by a deep neural network (DNN) with parameters θ:

$$\hat{u}(x, t; \theta) = \text{DNN}_\theta\big(\xi(x), \psi(t)\big) \approx u(x, t), \tag{6}$$

trained by minimizing the mean squared error (MSE) for initial conditions (IC), boundary conditions (BC), and interior data:

$$\mathcal{L}_{\text{IC}}(\theta) = \frac{1}{N_{\text{IC}}} \|\hat{u}_{\text{IC}} - u_{\text{IC}}\|_2^2, \tag{7}$$

$$\mathcal{L}_{\text{BC}}(\theta) = \frac{1}{N_{\text{BC}}} \|\hat{u}_{\text{BC}} - u_{\text{BC}}\|_2^2, \tag{8}$$

$$\mathcal{L}_{\text{Data}}(\theta) = \frac{1}{N_{\text{Data}}} \|\hat{u}_{\text{Data}} - u_{\text{Data}}\|_2^2, \tag{9}$$

where $\widehat{u}$ denotes the solution predicted by the DNN, u denotes the measured solution, and N_{IC}, N_{BC}, and N_{Data} are the number of points for each type of data. A warm-start solution is obtained by minimizing the weighted loss:

$$\mathcal{L}_{\text{warm}} = \lambda_{\text{IC}}\mathcal{L}_{\text{IC}} + \lambda_{\text{BC}}\mathcal{L}_{\text{BC}} + \lambda_{\text{Data}}\mathcal{L}_{\text{Data}} \tag{10}$$

where $\lambda_{\text{IC}}, \lambda_{\text{BC}}, \lambda_{\text{Data}}$ are either hyperparameters or dynamically adjusted during training to balance gradient magnitudes [21]:

$$\frac{\|\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\text{IC}}\|_2}{\|\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\text{Data}}\|_2}, \quad \frac{\|\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\text{BC}}\|_2}{\|\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\text{Data}}\|_2}. \tag{11}$$

Once a warm-start solution is obtained, the s candidate derivatives are computed via AD at the collocation points, forming the derivative library $\mathbf{D} \in \mathbb{R}^{N\times(s+1)}$, augmented with a column of ones. Sparsity is promoted via a derivative selection matrix $\boldsymbol{\Lambda} \in \mathbb{R}^{N\times(s+1)}$, parameterized by λ_i, obtained using sequential thresholded ridge regression (STRidge) [7, 18]. The product $\mathbf{D} \odot \boldsymbol{\Lambda}$ retains only active derivatives.

The coefficients of the derivatives are modeled by an MLP architecture $\text{MLP}_{\boldsymbol{\phi}}$, with options: 1) a single DNN with $s+1$ outputs, 2) a multi-task MLP with a shared backbone and $s+1$ shallow heads, or 3) a shallow NN for each coefficient. The choice depends on the expected correlation between coefficients.

The MLP outputs $s+1$ coefficients $f_0, \ldots, f_s$ at the collocation points, forming

$$\mathbf{F} \in \mathbb{R}^{N\times(s+1)}. \tag{12}$$

The physics-informed PDE loss is then

$$\mathcal{L}_{\text{PDE}}(\boldsymbol{\phi}) = \frac{1}{N_{\text{Col.}}}\left\|\hat{u}_{t,\text{Col.}} - (\mathbf{D}\odot\boldsymbol{\Lambda}\odot\mathbf{F})\mathbf{1}_{s+1}\right\|_2^2 \tag{13}$$

$$= \frac{1}{N_{\text{Col.}}}\left\|\hat{u}_{t,\text{Col.}} - \sum_{j=1}^{s+1}(\mathbf{D}\odot\boldsymbol{\Lambda})_{:,j}\odot\mathbf{F}_{:,j}\right\|_2^2. \tag{14}$$

Training proceeds in two stages. First, the solution network is optimized while freezing $\boldsymbol{\phi}$ and $\boldsymbol{\Lambda}$:

$$\mathcal{L}(\boldsymbol{\theta}) = \lambda_{\text{IC}}\mathcal{L}_{\text{IC}} + \lambda_{\text{BC}}\mathcal{L}_{\text{BC}} + \lambda_{\text{Data}}\mathcal{L}_{\text{Data}} + \lambda_{\text{PDE}}(\text{epoch})\mathcal{L}_{\text{PDE}}, \tag{15}$$

where $\lambda_{\text{IC}}, \lambda_{\text{BC}}, \lambda_{\text{Data}}$ terms are adjusted via gradient-balancing [21] and λ_{PDE} is slowly increases to 1 as a function of epoch.

Second, $\boldsymbol{\theta}$ is frozen and $\boldsymbol{\phi}, \boldsymbol{\Lambda}$ are optimized with STRidge using

$$\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\Lambda}) = \mathcal{L}_{\text{PDE}} + \beta\|\boldsymbol{\Lambda}\|_0, \tag{16}$$

where β controls sparsity.

4.2 Inference

Genetic Programming (GP) is an evolutionary algorithm designed to search the space of symbolic expressions and identify mathematical relationships that best fit data. In GP, candidate expressions are represented as syntax trees composed of mathematical operators, variables, and constants. These trees evolve iteratively through biologically inspired operations such as mutation, crossover, and selection. The evolution optimizes an objective function that balances predictive accuracy with model simplicity, enabling the flexible exploration of complex, nonlinear relationships without requiring a predefined functional form.

PySR (Python Symbolic Regression) [8] is a modern, scalable implementation of symbolic regression built on GP principles. PySR combines evolutionary search with gradient-based simplification to evolve concise symbolic expressions that fit input–output data $\mathbf{x} \mapsto y$, where $\mathbf{x}$ can be multidimensional. While performance degrades as the input dimension increases due to combinatorial explosion, PySR integrates numerical optimization and symbolic algebra to prune redundant terms, regularize constants, and simplify expressions during evolution. In our framework, PySR is used to infer closed-form expressions for the learned coefficient functions $f_0, f_1, \ldots, f_s$, which depend on (x, t, u). Available unary and binary operators are defined for the search, and each expression is assigned a numerical complexity, used to evaluate and rank candidate expressions.

After training, sparsity enforcement removes m inactive derivative terms, leaving $s' = s - m + 1$ active coefficients. For each active coefficient f_i, a dataset is constructed mapping the sampled inputs to the corresponding neural network–predicted coefficients:

$$(x, t, \widehat{u}) \mapsto f_i, \quad i \in \{0, \ldots, s'\}. \tag{17}$$

Each dataset is independently passed to PySR, which searches for symbolic expressions that best approximate the learned coefficients by minimizing the mean squared error (MSE). For each coefficient, we retain the top five expressions ranked by symbolic complexity and select the final expression through a combined analysis of loss, complexity, and domain-informed heuristics. This procedure yields interpretable, closed-form symbolic expressions for all active coefficients, representing the discovered PDE.

To guide the selection among candidate expressions, we use two measures. First, the MSE loss quantifies fit quality but can favor overly complex expressions. Second, a *score* balances fit and complexity by estimating the trade-off between loss reduction and increased complexity. The score for the ith expression is defined as

$$\text{score}_i = -\frac{\log(\text{loss}_i - \text{loss}_{i-1})}{\text{complexity}_i - \text{complexity}_{i-1}}. \tag{18}$$

Intuitively, the score compares the decrease in loss from the previous, simpler expression to the increase in complexity. The highest score corresponds to the most parsimonious expression that adequately fits the data, effectively tracking the Pareto front of simplicity versus accuracy.

One caveat arises when an expression consists solely of a constant, which is by definition the least complex model; such an expression has a default score of zero because there is no simpler reference expression. In these cases, the final selection is made manually by comparing both loss and complexity across candidate expressions.

5 DATA

Synthetic datasets for the PDEs listed in Table 2 were generated on uniformly discretized spatio-temporal grids using standard numerical differentiation and time-integration techniques.

For the viscous Burgers' equation,

$$u_t = -uu_x + \frac{0.01}{\pi}u_{xx}, \quad x \in [-1, 1],\ t \in [0, 1] \tag{19}$$

with initial and boundary conditions

$$u(x, 0) = -\sin(\pi x), \tag{20}$$

$$u(-1, t) = u(1, t) = 0, \tag{21}$$

we used the open-source MATLAB toolbox `Chebfun` [9], which represents functions on an interval using Chebyshev polynomial

Table 2: PDEs used to test Diff-SINDy.

Name	Equation
Homogeneous heat equation	$u_t = 0.005u_{xx}$
Wave equation	$u_{tt} = 1u_{xx}$
Burgers' equation	$u_t = -1uu_x + \frac{0.01}{\pi}u_{xx}$

expansions. The PDE was solved with the `pde23t` solver, which employs a second-order implicit time-stepping method. The solution was computed on a (256×100) (x, t) grid over $x \in [-1, 1]$ and $t \in [0, 1]$.

Datasets for the homogeneous heat and wave equations were generated using explicit finite-difference solvers [5, 6]. For the heat equation,

$$u_t = 0.005u_{xx}, \quad x \in [-1, 1],\ t \in [0, 10] \tag{22}$$

with

$$u(x, 0) = 50, \tag{23}$$
$$u(-1, t) = 70, \tag{24}$$
$$u(1, t) = 90, \tag{25}$$

the second spatial derivative was approximated using a second-order central finite difference, and time integration was performed via a forward Euler scheme. The solution was computed on a (100×300) (x, t) grid.

For the wave equation,

$$u_{tt} = u_{xx}, \quad x \in [-1, 1],\ t \in [0, 1] \tag{26}$$

with

$$u(x, 0) = \sin(2\pi x), \tag{27}$$
$$u(-1, t) = u(1, t) = 0, \tag{28}$$

an explicit finite-difference time-stepping method was applied with a standard second-order spatial discretization. The solution was computed on a (100×300) (x, t) grid with a zero-velocity sine-wave initial condition and homogeneous Dirichlet boundary conditions.

6 EXPERIMENTS

In this section, we describe the experiments conducted using our proposed Diff-SINDy pipeline and highlight aspects that remain future work. We applied our method to re-discover three PDEs: the homogeneous heat equation, Burgers' equation, and the wave equation listed in Table 2 and described in Section 5. Where available, we compare our results to PINN-SR [7] and Evo-SINDy [11] in Table 6, noting that they were applied to different datasets. For all experiments, we used noise-free data and provided a derivative library containing only the known active terms; sparsity enforcement will be incorporated in future work.

For symbolic regression (SR), we provided the same operator set for all three PDEs: unary operators `[square, cube, exp, cos, sin]` and binary operators `[*, +, -, /]`. Complexity assignments were as follows: constants and unary operators 1 (except / with complexity 2), `cos` and `sin` complexity 2, and `exp` complexity 4. Slight variations in complexity did not change the top-scoring expressions. Early stopping was implemented when MSE $< 10^{-10}$ and complexity < 10 to promote parsimonious solutions.

All experiments were performed on Google Colab using a single T4 GPU, with training taking approximately 30 minutes and inference 10 minutes. We generated 10,000 collocation points per experiment using dense Latin Hypercube sampling.

6.1 Homogeneous heat equation

The heat equation dataset consisted of 30,000 (x, t) pairs, with 100, 600, and 29,300 IC, BC, and interior points, respectively. A single MLP with four hidden layers of dimension 512 and GELU activations was used to approximate the solution, with a 200-epoch warm start. The derivative library contained only u_{xx}, and a single MLP with identical architecture was used to predict its coefficient. Two-stage training was performed over 1,000 epochs using the Adam optimizer (learning rate 10^{-4}), with λ_{PDE} increasing from 0.18 to 1 at epoch 500.

Table 3 lists the top five symbolic expressions discovered by PySR. The simplest model, 0.00494, had an MSE of 3.871×10^{-6}, representing only a 1% error relative to the true coefficient (0.005). Despite having a score of 0 by definition, it was chosen as the most parsimonious expression because the lowest-MSE expression achieved only a marginal improvement while being nine times more complex.

Table 3: PySR results for the coefficient of u_{xx} in the homogeneous heat equation (true coefficient 0.005).

Complexity	Expression	MSE $[10^{-6}]$	Score
1	0.00494	3.871	0.000
5	$-0.0006\,x + 0.0047$	3.761	0.007
6	$0.0024\,x^2 + 0.0040$	3.392	0.103
7	$-0.0014\cos(u) + 0.006$	2.889	0.160
9	$-0.001\cos(0.38\,u) + 0.005$	2.238	0.128

6.2 Wave equation

The wave equation dataset contained 30,000 (x, t) points, with 100, 600, and 29,300 IC, BC, and interior points. A single MLP with four hidden layers of dimension 512 and GELU activations was used, with a 200-epoch warm start. The derivative library included only u_{xx}, predicted with a single MLP of the same architecture. Training followed the same two-stage procedure as above.

Table 4 shows the top five expressions for the u_{xx} coefficient. The simplest model, 0.997, achieved an MSE of 1.652×10^{-6}, corresponding to 0.3% error. Although its score is 0, it was selected for its parsimony, as more complex expressions offered only minor improvements.

6.3 Burgers' equation

The Burgers' equation dataset contained 25,600 (x, t) pairs, with 256, 200, and 25,146 IC, BC, and interior points. A multi-task MLP with a shared backbone of four hidden layers (dimension 512) and two-heads (two hidden layers per head, dimension 512) was used to predict the coefficients of u_x and u_{xx}. A 200-epoch warm start was provided, and training followed the two-stage procedure with

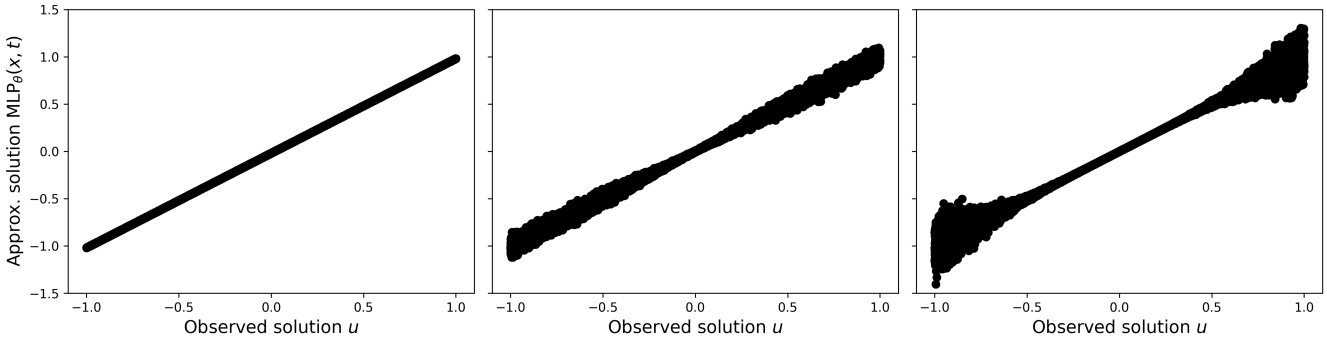


Figure 2: Predicted solutions from three differently initialized (with different architectures and settings for training) solution MLPs for Burgers' equation. Despite differences in solution accuracy, PySR recovered the same symbolic expressions (Table 5) with minor variations in MSE and score.

Table 4: PySR results for the coefficient of u_{xx} in the wave equation (true coefficient 1).

Complexity	Expression	MSE $[10^{-6}]$	Score
1	0.997	1.652	0.000
5	$\cos(0.116\,x)$	1.563	0.238
6	$-0.006\,x^2 + 0.999$	1.356	0.141
8	$-0.006(x + 0.006)^2 + 0.999$	1.355	0.001
10	$-0.006\,x^2\cos(u) + 0.999$	1.165	0.008

Adam optimizer (learning rate 10^{-4}), λ_{PDE} increasing from 0.18 to 1 at epoch 500.

Table 5 lists the top five PySR expressions for both coefficients. For u_x, the second simplest expression $-0.995\,u$ achieved the highest score (3.346) and an MSE of 3.451×10^{-4}, with only 0.5% error relative to the true coefficient $-u$. For u_{xx}, the simplest expression 0.00298 was selected with a 6% error relative to $0.01/\pi$.

Table 5: PySR results for the coefficients of u_x and u_{xx} in Burgers' equation (true coefficients $-u$ and $0.01/\pi \approx 0.00318$, respectively).

Complexity	Expression	MSE	Score
u_x coefficient			
1	x	0.2787	0.000
3	$-0.995\,u$	3.451×10^{-4}	3.346
5	$-0.982\,u + 0.003$	3.392×10^{-4}	0.0099
7	$-0.958(-0.028\,x + u)$	1.835×10^{-4}	0.510
10	$-0.966\,u + 0.032(x + 0.138)^3$	1.624×10^{-4}	0.085
u_{xx} coefficient			
1	0.00298	2.148×10^{-7}	0.000
5	$0.0009x + 0.003$	1.962×10^{-7}	0.242
7	$0.004\cos(x - 0.303)$	1.334×10^{-7}	0.191
8	$0.0005(x - 0.668)^3 + 0.003$	9.729×10^{-8}	0.316
10	$(0.006\,u + 0.001)x^3 + 0.003$	8.011×10^{-8}	0.097

6.4 Robustness to noise

While we have not explicitly added noise in these experiments, we provide an illustration of robustness using Burgers' equation. Figure 2 shows three solution MLPs trained with different initializations. Although the predicted solutions vary, the same PySR expressions were recovered, with slight variations in MSE and score, indicating that our symbolic regression framework is relatively robust to solution differences arising from model initialization or introduced by noise.

Table 6: Comparison for relative error (with respect to the true value) in coefficients between PINN-SR [7], Evo-SINDy [11], and Diff-SINDy. For Burgers' equation we state the error in the u_x/u_{xx} coefficients.

PDE	PINN-SR	Evo-SINDy	**Diff-SINDy**
Homogeneous heat eq.	—	—	1%
Wave eq.	—	0.01%	0.3%
Burgers' eq.	0.9%/1%	0%/2.3%	0.5%/6%

7 Future Works

Diff-SINDy is still under active development, and we aim to expand its functionality in several key directions. First, we plan to incorporate soft-thresholding–based sparsity with improved mechanisms for training stability. We also intend to broaden the set of architectures used for both solution and coefficient approximation, including transformer-based models. In addition, we will extend our experiments to higher-dimensional PDEs and work toward automating the discovery of constant coefficients in settings where a score measure is not available. We will further explore how pretraining on families of PDEs may improve generalization, convergence, and computational efficiency. Finally, we will examine the robustness of our results under noise injected into both the input coordinates and observed solutions, and study how performance scales as the size of the derivative library increases.

A further important direction is the enforcement of physical structure within the learned symbolic expressions. For instance, while the wave equation satisfies conservation of energy, many candidate expressions may violate this principle. We aim to eliminate such unphysical solutions by incorporating prior knowledge—such

as conservation laws and other physical constraints—directly into the training and inference pipelines.

8 Conclusion

We introduced Diff-SINDy, a fully differentiable framework for parsimonious PDE discovery that promotes sparsity through soft-thresholding. Diff-SINDy replaces the large functional libraries typical of equation-learning methods with a compact derivative library that leverages the inherent structure and simplicity of physical PDEs. The framework cleanly separates (i) the training of the approximate solution and coefficient fields from (ii) the inference of symbolic coefficient expressions. Because all components remain differentiable, derivatives are computed efficiently through automatic differentiation. The method is computationally inexpensive, requires no pretraining, and scales naturally with problem complexity.

We validated Diff-SINDy on the homogeneous heat, wave, and Burgers' equations, where it successfully rediscovered the governing PDEs using a scoring objective that balances accuracy with symbolic complexity. When such a score is available, Diff-SINDy selects expressions along this trade-off curve; when it is not, we employ heuristics that compare reductions in loss to increases in expression complexity. These experiments demonstrate the potential of Diff-SINDy to recover physically meaningful laws from data while avoiding unnecessary complexity.

Looking forward, we aim to further improve the robustness and stability of the framework, particularly in the presence of noise and in settings with highly nonconvex loss landscapes. Additional directions include exploring pretraining on families of PDEs, expanding architectural choices for solution and coefficient approximation, and integrating physical constraints—such as conservation laws—directly into the training and inference pipelines. Together, these efforts will advance Diff-SINDy toward a more versatile and reliable tool for data-driven scientific discovery.

9 DATA and CODE AVAILABILITY

The code for Diff-SINDy and the data for the experiments can be found on EA's github `https://github.com/ealjamal/Diff-SINDy` and website https://ealjamal.github.io/Diff-SINDy/.

References

[1] Steven Atkinson, Waad Subber, Liping Wang, Genghis Khan, Philippe Hawi, and Roger Ghanem. 2019. Data-driven discovery of free-form governing differential equations. *arXiv e-prints*, Article arXiv:1910.05117 (Sept. 2019), arXiv:1910.05117 pages. arXiv:1910.05117 [cs.CE] doi:10.48550/arXiv.1910.05117

[2] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. 2021. Neural Symbolic Regression that Scales. *arXiv e-prints*, Article arXiv:2106.06427 (June 2021), arXiv:2106.06427 pages. arXiv:2106.06427 [cs.LG] doi:10.48550/arXiv.2106.06427

[3] Josh Bongard and Hod Lipson. 2007. From the Cover: Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Science* 104, 24 (June 2007), 9943–9948. doi:10.1073/pnas.0609476104

[4] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. 2016. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Science* 113, 15 (April 2016), 3932–3937. arXiv:1509.03580 [math.DS] doi:10.1073/pnas.1517384113

[5] John Burkardt. 2012. Finite Difference Method for the 1D Wave Equation. https://people.math.sc.edu/Burkardt/cpp_src/fd1d_wave/fd1d_wave.html.

[6] John Burkardt. 2014. Finite Difference Solution of the Time Dependent 1D Heat Equation using Explicit Time Stepping. https://people.math.sc.edu/Burkardt/py_src/fd1d_heat_explicit/fd1d_heat_explicit.html.

[7] Zhao Chen, Yang Liu, and Hao Sun. 2021. Physics-informed learning of governing equations from scarce data. *Nature Communications* 12, Article 6136 (Oct. 2021), 6136 pages. arXiv:2005.03448 [cs.LG] doi:10.1038/s41467-021-26434-1

[8] Miles Cranmer. 2023. Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl. *arXiv e-prints*, Article arXiv:2305.01582 (May 2023), arXiv:2305.01582 pages. arXiv:2305.01582 [astro-ph.IM] doi:10.48550/arXiv.2305.01582

[9] T. A. Driscoll, N. Hale, and L. N. Trefethen (Eds.). 2014. *Chebfun Guide*. Pafnuty Publications, Oxford.

[10] U Fasel, J. N. Kutz, B. W. Brunton, and Brunton S. L. 2022. Ensemble-SINDy: Robust sparse model discovery in the low-data, high-noise limit, with active learning and control. *Proceedings of the Royal Society A* 478, 2260 (June 2022), 9943–9948. doi:10.1098/rspa.2021.0904

[11] Yuxin Jiang and Jianyong Sun. 2025. Evo-SINDy: Universal Discovery of Partial Differential Equations Using Cooperative Evolutionary Computation. In *Proceedings of the Genetic and Evolutionary Computation Conference* (NH Malaga Hotel, Malaga, Spain) *(GECCO '25)*. Association for Computing Machinery, New York, NY, USA, 791–799. doi:10.1145/3712256.3726360

[12] Krzysztof Kacprzyk, Zhaozhi Qian, and Mihaela van der Schaar. 2023. D-CIPHER: Discovery of Closed-form Partial Differential Equations. In *Thirty-seventh Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=jnCPN1vpSR

[13] Kadierdan Kaheman, Steven L Brunton, and J Nathan Kutz. 2022. Automatic differentiation to simultaneously identify nonlinear dynamics and extract noise probability distributions from data. *Machine Learning: Science and Technology* 3, 1 (mar 2022), 015031. doi:10.1088/2632-2153/ac567a

[14] Michail Maslyaev, Alexander Hvatov, and Anna Kalyuzhnaya. 2019. Data-driven PDE discovery with evolutionary approach. *arXiv e-prints*, Article arXiv:1903.08011 (March 2019), arXiv:1903.08011 pages. arXiv:1903.08011 [cs.NE] doi:10.48550/arXiv.1903.08011

[15] Zhaozhi Qian, Krzysztof Kacprzyk, and Mihaela van der Schaar. 2022. D-CODE: Discovering Closed-form ODEs from Observed Trajectories. In *International Conference on Learning Representations*. https://openreview.net/forum?id=wENMvIsxNN

[16] Maziar Raissi, Paris Perdikaris, Nazanin Ahmadi, and George Em Karniadakis. 2024. Physics-Informed Neural Networks and Extensions. *arXiv e-prints*, Article arXiv:2408.16806 (Aug. 2024), arXiv:2408.16806 pages. arXiv:2408.16806 [cs.LG] doi:10.48550/arXiv.2408.16806

[17] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. 2017. Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations. *arXiv e-prints*, Article arXiv:1711.10561 (Nov. 2017), arXiv:1711.10561 pages. arXiv:1711.10561 [cs.AI] doi:10.48550/arXiv.1711.10561

[18] Samuel H. Rudy, Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. 2017. Data-driven discovery of partial differential equations. *Science Advances* 3, 4, Article e1602614 (April 2017), e1602614 pages. arXiv:1609.06401 [nlin.PS] doi:10.1126/sciadv.1602614

[19] Michael Schmidt and Hod Lipson. 2009. Distilling Free-Form Natural Laws from Experimental Data. *Science* 324, 5923 (April 2009), 81. doi:10.1126/science.1165893

[20] Parshin Shojaee, Kazem Meidani, Amir Barati Farimani, and Chandan K. Reddy. 2023. Transformer-based Planning for Symbolic Regression. *arXiv e-prints*, Article arXiv:2303.06833 (March 2023), arXiv:2303.06833 pages. arXiv:2303.06833 [cs.LG] doi:10.48550/arXiv.2303.06833

[21] Sifan Wang, Yujun Teng, and Paris Perdikaris. 2021. Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks. *SIAM Journal on Scientific Computing* 43, 5 (Jan. 2021), A3055–A3081. arXiv:2001.04536 [cs.LG] doi:10.1137/20M1318043

[22] Hao Xu, Haibin Chang, and Dongxiao Zhang. 2020. DLGA-PDE: Discovery of PDEs with incomplete candidate library via combination of deep learning and genetic algorithm. *J. Comput. Phys.* 418 (2020), 109584. doi:10.1016/j.jcp.2020.109584