

# Automatic Solution of Partial/Ordinary Differential Equations with Physics-Informed Neural Networks

Shirshendu Sekhar Mondal

December 21, 2025

## Abstract

This work presents an end-to-end framework for solving partial differential equations (PDEs) and ordinary differential equations (ODEs) starting directly from their human-readable representation. The system accepts equations written in text, L<sup>A</sup>T<sub>E</sub>X, or extracted from images, converts them into a symbolic residual form, and routes the problem either to a classical symbolic solver or to a Physics-Informed Neural Network (PINN). For the PINN backend, the method is demonstrated on the one-dimensional viscous Burgers equation with viscosity  $\nu = 0.01$  on a bounded space-time domain. The framework produces both numerical solution grids and quantitative residual diagnostics, enabling systematic assessment of solution quality and serving as a reproducible research tool for scientific machine learning.

## 1 Introduction

Many problems in physics and engineering are formulated as partial differential equations (PDEs), but moving from a written equation to a numerical or analytic solution usually requires several manual steps: rewriting the equation for a chosen library, implementing boundary and initial conditions, and constructing a dedicated numerical solver. This workflow is time-consuming and error-prone, especially when exploring many model variants or parameter regimes. Physics-Informed Neural Networks (PINNs) have emerged as a flexible alternative, as they approximate the solution of a PDE by training a neural network to satisfy the governing equation and associated constraints without requiring labeled solution data.

The goal of this project is to integrate equation parsing, symbolic manipulation, solver selection, and PINN training into a single, configuration-driven pipeline. A user can supply a PDE in L<sup>A</sup>T<sub>E</sub>X or plain text, specify problem metadata in a YAML configuration file, and obtain either an analytic solution (when available) or a PINN-based numerical approximation, together with diagnostics and visualizations suitable for analysis and publication.

## 2 Problem Formulation

### 2.1 PDE and domain

The framework is designed to operate on general PDEs that can be expressed in residual form

$$\mathcal{F}(u, \partial u, \partial^2 u, \dots; \boldsymbol{x}) = 0, \quad (1)$$

where  $u$  is the unknown field,  $\boldsymbol{x}$  denotes the independent variables, and  $\mathcal{F}$  is a differential operator. For the case study in this report, the PDE is the one-dimensional viscous Burgers equation with viscosity  $\nu = 0.01$ :

$$u_t + u u_x - \nu u_{xx} = 0, \quad (x, t) \in [-1, 1] \times [0, 1]. \quad (2)$$

The initial condition is given by

$$u(x, 0) = -\sin(\pi x), \quad x \in [-1, 1], \quad (3)$$

and homogeneous Dirichlet boundary conditions are imposed at the spatial boundaries,

$$u(-1, t) = 0, \quad u(1, t) = 0, \quad t \in [0, 1]. \quad (4)$$

These conditions define a well-posed forward problem that is widely used as a benchmark in the PINN literature due to the development of steep gradients and near-shock structures for small viscosity.

## 2.2 Symbolic residual form

Internally, the PDE (2) is rewritten into a residual form

$$R(x, t; u) = u_t + u u_x - \nu u_{xx}, \quad (5)$$

so that an exact solution satisfies  $R(x, t; u) = 0$  everywhere on the domain. This residual formulation is common across the entire framework: it is used by the symbolic solver, by the PINN loss function, and by post-processing diagnostics that quantify the degree to which the learned solution satisfies the PDE.

## 3 Methodology

### 3.1 Equation ingestion and parsing

The front end of the system accepts PDEs from multiple sources:

- **Text and L<sup>A</sup>T<sub>E</sub>X files:** the equation is read from a `.txt` or `.tex` file and parsed into a symbolic expression.
- **Images:** an equation written on paper or in a screenshot can be processed by an OCR-style step and converted into the same symbolic representation.

In all cases, the resulting expression is normalized into a canonical residual form. This normalization separates the problem definition from the choice of numerical method and enables the same PDE to be solved by different backends with minimal additional effort.

### 3.2 Configuration-based problem specification

A YAML configuration file specifies all non-equation meta

- the list of independent variables (e.g. `[x, t]`);
- domain bounds for each variable;
- physical constants such as the viscosity  $\nu$ ;
- initial and boundary conditions, described by samplers and targets;
- hyperparameters for PINN training (number of collocation points, batch sizes, optimizer settings, and loss weights);
- output settings such as grid resolution and filenames for saved arrays and plots.

This design makes experiments easily reproducible: changing the problem or the training regime only requires edits to the configuration file, while the core pipeline remains unchanged.

### 3.3 Solver routing

Once the equation and configuration are loaded, a routing component selects an appropriate solver:

- **Symbolic solver:** for simpler or lower-dimensional equations, the system attempts to obtain closed-form solutions using a computer algebra system. If successful, the analytic expression and associated solver hints are returned.
- **PINN solver:** for more complex PDEs, or when explicitly requested by the user, the problem is handed to the PINN backend. The router constructs the domain object, the condition objects (for IC and BC), and a set of training hyperparameters that are passed to the PINN training routine.

## 4 Physics-Informed Neural Network

### 4.1 Network structure

In the PINN approach, the unknown solution  $u(x, t)$  is approximated by a neural network  $u_\theta(x, t)$  with parameters  $\theta$ . The network takes  $(x, t)$  as input and returns a scalar output interpreted as the solution value at that point. Smooth activation functions, such as tanh, are typically employed to facilitate the computation of higher-order derivatives via automatic differentiation.

### 4.2 Loss function

The training objective combines a PDE residual loss with losses that enforce initial and boundary conditions. Let  $\{(x_i, t_i)\}_{i=1}^{N_f}$  be collocation points sampled in the interior of the domain, and let  $\{(\bar{x}_j, \bar{t}_j)\}_{j=1}^{N_c}$  be points sampled on the IC/BC manifolds. The residual loss is defined as

$$\mathcal{L}_f(\theta) = \frac{1}{N_f} \sum_{i=1}^{N_f} |R(x_i, t_i; u_\theta)|^2, \quad (6)$$

where the residual  $R$  uses derivatives of  $u_\theta$  computed through automatic differentiation. The constraint loss collects the discrepancy between the network output and the prescribed targets,

$$\mathcal{L}_c(\theta) = \frac{1}{N_c} \sum_{j=1}^{N_c} |u_\theta(\bar{x}_j, \bar{t}_j) - u_{\text{target}}(\bar{x}_j, \bar{t}_j)|^2. \quad (7)$$

To balance the contributions from the PDE and the constraints, the total loss is formed as

$$\mathcal{L}(\theta) = w_f \mathcal{L}_f(\theta) + w_c \mathcal{L}_c(\theta), \quad (8)$$

where  $w_f$  and  $w_c$  are user-specified weights. For the Burgers problem considered here, interior residuals and boundary/initial conditions are given comparable importance by setting  $w_f = 1$  and  $w_c = 10$  in the configuration.

### 4.3 Optimization strategy

Training is performed in two phases. First, the Adam optimizer is used for a prescribed number of iterations to explore the loss surface and obtain a reasonable initial approximation. Second, the parameters are refined using the L-BFGS optimizer, which is a quasi-Newton method suitable for smooth objectives. During L-BFGS, a closure function recomputes the loss and gradients at each step, allowing the optimizer to build curvature information and converge to a lower-loss solution.

## 5 Implementation

### 5.1 Software architecture

The implementation is organized into modular components:

- **CLI interface:** a command-line entry point exposes options for specifying the equation file, input type (text, L<sup>A</sup>T<sub>E</sub>X, or image), preferred solver, and configuration path.
- **Equation parser:** functions that read and interpret the input equation, returning an internal symbolic representation.
- **Configuration loader:** utilities that parse the YAML configuration and construct domain and condition objects.
- **Router:** logic that chooses between symbolic and PINN solvers and aggregates results.
- **PINN solver:** code that implements sampling of collocation and constraint points, network construction, loss evaluation, and optimization.
- **Post-processing:** routines that evaluate the trained network on uniform grids, save NumPy arrays, create visualizations, and compute residual statistics.

### 5.2 Sampling and evaluation

For training, interior collocation points are drawn uniformly in the rectangular domain  $[-1, 1] \times [0, 1]$ . Points for the initial condition are sampled along the line  $t = 0$ , and boundary points are drawn on  $x = -1$  and  $x = 1$ . After training, the network is evaluated on a Cartesian grid in  $(x, t)$  with user-specified resolution. The resulting values are saved as a NumPy array for further analysis and plotted as a heatmap.

Additionally, a large number of random points in the domain are used to estimate residual statistics, including the mean residual, mean absolute error (MAE), root-mean-square error (RMSE), and maximum absolute residual.

## 6 Results

### 6.1 PINN solution of Burgers' equation

Figure 1 shows the learned solution  $u_\theta(x, t)$  of the viscous Burgers equation on the space-time domain. The horizontal axis represents the spatial coordinate  $x \in [-1, 1]$ , the vertical axis represents time  $t \in [0, 1]$ , and the color encodes the magnitude of the solution field.

The initial condition  $u(x, 0) = -\sin(\pi x)$  is clearly visible at the bottom of the plot: the solution starts as a smooth sine wave that is antisymmetric with respect to the origin. As time progresses, nonlinear advection and diffusion interact, leading to the formation of a traveling wave structure that propagates through the domain while gradually smoothing due to viscosity. The boundary conditions  $u(-1, t) = u(1, t) = 0$  confine the dynamics and are enforced by the constraint part of the loss.

The current training configuration yields residual statistics with a small but non-zero discrepancy between the learned solution and exact PDE satisfaction. The residual root-mean-square error is approximately of order  $10^{-2}$ , and the maximum absolute residual on a large random sample of domain points remains below  $10^{-1}$ . These values indicate that the PINN has captured the overall qualitative behavior and principal features of the viscous Burgers solution, while leaving room for further refinement through additional training, adaptive sampling, or modified loss weighting.

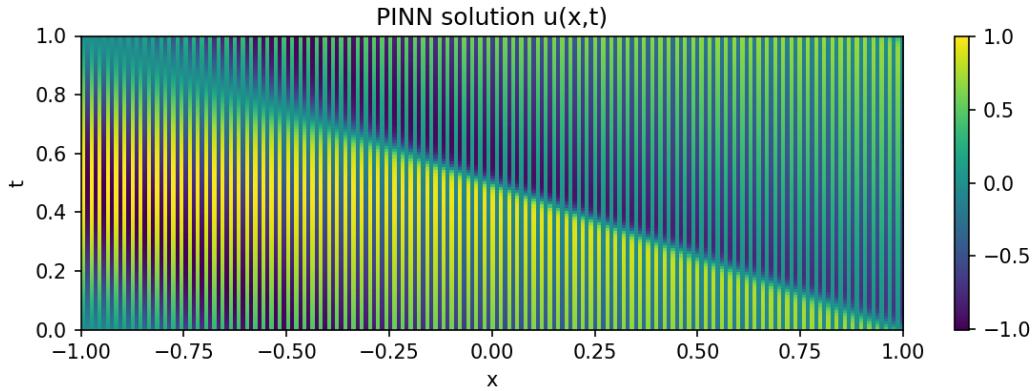


Figure 1: PINN approximation of the solution  $u(x, t)$  for the viscous Burgers equation with  $\nu = 0.01$ . The plot is obtained by evaluating the trained network on a uniform grid in  $(x, t)$  and visualizing the result as a heatmap.

## 6.2 Discussion

The results highlight both the strengths and limitations of the present pipeline. On the one hand, the framework can automatically ingest a written PDE, configure a PINN, and produce a plausible space-time solution without manually coding a problem-specific numerical scheme. On the other hand, the residual diagnostics and the fine-scale structures in Figure 1 suggest that accurately resolving sharp gradients for low-viscosity Burgers equations remains challenging, and that advanced techniques such as adaptive weighting, residual-based sampling, or specialized network architectures may be beneficial in future work.

## 7 Conclusion

This project demonstrates an integrated approach to solving PDEs that begins with a human-readable equation and ends with both analytic or PINN-based numerical solutions. By combining symbolic parsing, configuration-driven problem specification, automated solver routing, and a robust PINN training loop, the framework reduces the barrier between mathematical models and computational experiments. The Burgers equation example illustrates how the system can approximate complex, nonlinear dynamics and provide quantitative measures of fidelity to the underlying physics.

Future extensions could include support for systems of PDEs, irregular geometries, parameter inference problems, and adaptive strategies that refine the sampling or the loss weights based on residual information. Such developments would further enhance the utility of the framework for scientific computing, inverse problems, and data-free model exploration.