Solving Nonlinear Partial Differential Equations Using Physics-Informed Neural Networks

BTech Project 3rd Sem Report

Ramagiri Jaidhar
EP23BTECH11021

భారతీయ సాంకేతిక విజ్ఞాన సంస్థ హైదరాబాద్
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Department of Physics
Supervisor: Dr. Kirit Makwana

Abstract

Physics-Informed Neural Networks (PINNs) offer a modern way to solve partial differential equations (PDEs) by weaving the physical laws directly into the structure of a neural network. In this project, I apply PINNs to several nonlinear systems—the Burgers equation, the Riemann problem, the three-dimensional heat equation, and a kinematic dynamo model. Each case probes a different challenge: nonlinear evolution, sharp discontinuities, high-dimensional inputs, or coupled vector fields. Across the smooth problems, the networks learn the underlying physics well and reproduce expected solution behavior without relying on traditional discretization. At the same time, the experiments show clear limits: fully connected PINNs have trouble capturing shocks and face significant difficulty when dealing with high-dimensional, curl-driven vector equations. These results highlight both the promise of PINNs and the areas where more expressive architectures are needed.

1 Introduction

Partial Differential Equations (PDEs) are fundamental in describing various physical systems such as fluid flow, heat transfer, and electromagnetism. Traditional numerical methods like finite difference or finite element schemes rely on discretization, which becomes computationally expensive for high-dimensional or complex geometries. Physics-Informed Neural Networks (PINNs) offer a mesh-free alternative by incorporating governing equations directly into the neural network loss function.

The objective of this project is to implement and analyze PINNs to solve different nonlinear PDEs of increasing complexity. Starting with the Burgers equation as a benchmark, the study progresses to the Riemann problem, the 3D heat equation, and finally attempts a kinematic dynamo model to test vector-field learning capabilities.

2 Theoretical Background

This section outlines the essential mathematical ingredients used in Physics-Informed Neural Networks (PINNs) for solving nonlinear partial differential equations.

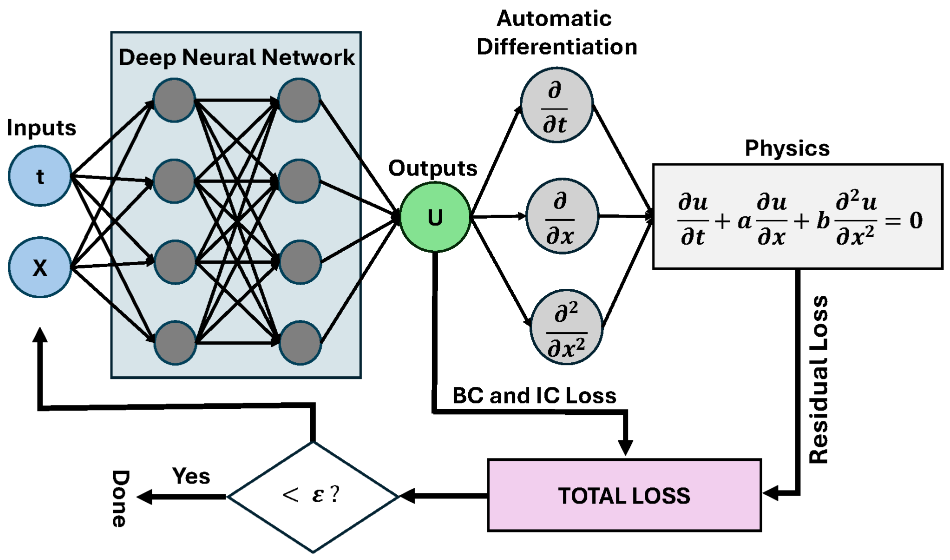

Deep Neural Network
Automatic Differentiation
Inputs
t
X
Outputs
U
$\frac{\partial}{\partial t}$
$\frac{\partial}{\partial x}$
$\frac{\partial^2}{\partial x^2}$
Physics
$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} + b\frac{\partial^2 u}{\partial x^2} = 0$
Residual Loss
BC and IC Loss
TOTAL LOSS
$< \varepsilon$?
Yes
Done

Figure 1: General PINN architecture.

2.1 Deep Neural Networks

Definition. Let $\mathbf{x} = (x_1, \ldots, x_n)^\top \in \mathbb{R}^n$ be the input vector. A single neuron with weights $\mathbf{w} \in \mathbb{R}^n$ and bias $b \in \mathbb{R}$ computes the pre-activation

$$z = \mathbf{w}^\top \mathbf{x} + b,$$

and an activation $\sigma : \mathbb{R} \to \mathbb{R}$ produces

$$u(\mathbf{x}) = \sigma(z).$$

For a fully connected feed-forward neural network with L layers, denote the layer outputs $\mathbf{z}^{(l)}$. Then:

$$\mathbf{z}^{(1)}(\mathbf{x},t) = \mathbf{x}_0(\mathbf{x},t) \in \mathbb{R}^{n_1}, \tag{1}$$

$$\mathbf{z}^{(l)} = \sigma\Big(W^{(l-1)}\mathbf{z}^{(l-1)} + \mathbf{b}^{(l-1)}\Big), \qquad l = 2,\dots,L-1, \tag{2}$$

$$\mathbf{z}^{(L)} = W^{(L-1)}\mathbf{z}^{(L-1)} + \mathbf{b}^{(L-1)} \in \mathbb{R}^{n_{\text{out}}}. \tag{3}$$

Typical activations include tanh, Sigmoid, ReLU, and Leaky ReLU. The fully connected structure allows the network to act as a universal approximator for sufficiently smooth targets.

2.2 PINNs for Initial/Boundary Value Problems

Let $\Omega \subset \mathbb{R}^d$ be a compact spatial domain and $t \in [0,T]$ the temporal interval. Consider a general nonlinear PDE written as

$$\mathcal{N}\big[u(\mathbf{x},t)\big] = 0, \qquad \mathbf{x} \in \Omega,\ t \in (0,T], \tag{4}$$

with initial condition

$$u(\mathbf{x},0) = h(\mathbf{x}), \qquad \mathbf{x} \in \Omega, \tag{5}$$

and boundary condition

$$\mathcal{B}[u(\mathbf{x},t)] = g(\mathbf{x},t), \qquad \mathbf{x} \in \partial\Omega,\ t \in [0,T]. \tag{6}$$

In a PINN we approximate $u(\mathbf{x},t)$ by a neural network $u_\theta(\mathbf{x},t)$, parameterized by θ. Automatic differentiation computes derivatives that appear in $\mathcal{N}$, e.g. $\partial_t u_\theta,\ \partial_{x_i} u_\theta,\ \partial_{x_i x_j} u_\theta$.

Define the PDE residual

$$r_\theta(\mathbf{x},t) = \mathcal{N}\big[u_\theta(\mathbf{x},t)\big].$$

2.3 Composite Loss Function

Training minimizes a composite loss enforcing the PDE, initial condition, and boundary condition. Following the standard empirical formulation, the total loss is

$$\mathcal{L}_{\text{total}}(\theta) = \lambda_{\text{PDE}}\,\mathcal{L}_{\text{PDE}}(\theta) + \lambda_{\text{IC}}\,\mathcal{L}_{\text{IC}}(\theta) + \lambda_{\text{BC}}\,\mathcal{L}_{\text{BC}}(\theta). \tag{7}$$

Using discrete sets of collocation points:

- Residual points $\{(\mathbf{x}_i,t_i)\}_{i=1}^{N_r}$ inside $\Omega \times (0,T]$,
- Initial points $\{\mathbf{x}_j\}_{j=1}^{N_0}$ (at $t=0$),
- Boundary points $\{(\mathbf{x}_k,t_k)\}_{k=1}^{N_b}$ on $\partial\Omega \times [0,T]$,

we define

$$\mathcal{L}_{\text{PDE}}(\theta) = \frac{1}{N_r}\sum_{i=1}^{N_r}\big|r_\theta(\mathbf{x}_i,t_i)\big|^2, \tag{8}$$

$$\mathcal{L}_{\text{IC}}(\theta) = \frac{1}{N_0}\sum_{j=1}^{N_0}\big|u_\theta(\mathbf{x}_j,0) - h(\mathbf{x}_j)\big|^2, \tag{9}$$

$$\mathcal{L}_{\text{BC}}(\theta) = \frac{1}{N_b}\sum_{k=1}^{N_b}\big|\mathcal{B}[u_\theta](\mathbf{x}_k,t_k) - g(\mathbf{x}_k,t_k)\big|^2. \tag{10}$$

Optimizers such as Adam (for initial training) and L-BFGS (for refinement) are commonly applied to minimize $\mathcal{L}_{\text{total}}$.

3 Methodology

All models in this project were implemented in PyTorch and trained on a GPU. Each Physics-Informed Neural Network (PINN) was built from fully connected layers, with activation functions chosen based on the type of PDE being solved. The networks received spatial and temporal coordinates as inputs and produced either scalar or vector fields as outputs, depending on the physical system.

To guide the network toward physically meaningful solutions, the total loss combined the PDE residual, the initial conditions, and the boundary conditions through appropriate weighting. Using automatic differentiation, PyTorch computed all required derivatives directly from the network outputs, making it possible to evaluate the PDE residual at any collocation point.

Training was performed in two stages. The first stage used the Adam optimizer to explore the loss landscape and make rapid progress toward a reasonable solution. The second stage refined the model using the L–BFGS optimizer, which helped improve accuracy and smoothness once the network was already close to a minimum.

4 Case Studies and Results

4.1 Burgers Equation

The one-dimensional viscous Burgers equation,

$$u_t + u\,u_x = \nu\,u_{xx}, \tag{11}$$

is a classic testbed for Physics-Informed Neural Networks. Although the equation looks simple, it blends nonlinear advection with diffusion, producing solutions that start smooth but quickly develop steep gradients and shock-like structures. Because of this behavior, Burgers' equation is widely used to check whether a PINN can correctly learn both first- and second-order derivatives through automatic differentiation.

It is also one of the main examples used in the original PINN work by Raissi, Perdikaris, and Karniadakis. Their publicly available implementation (`https://github.com/maziarraissi/PINNs/`) has become a standard reference for benchmarking new PINN models. Part of the motivation for including Burgers' equation in our project is to verify that our PINN behaves consistently with this original model and reproduces its characteristic solution patterns.

Problem Setup. We consider the domain $x \in [-1, 1]$ and $t \in [0, 2.5]$. The initial condition

$$u(x, 0) = -\sin(\pi x)$$

generates a wave that steepens over time. At the boundaries we impose

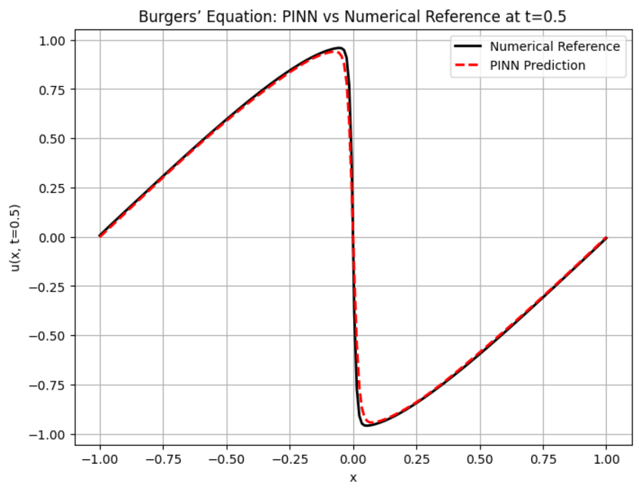
$$u(-1, t) = 0, \qquad u(1, t) = 0,$$

and set the viscosity to

$$\nu = \frac{0.01}{\pi},$$

Training Strategy. Our neural network takes (x, t) as input and outputs $u(x, t)$. Training relies on three sets of points: 200 initial-condition points along $t = 0$, 200 boundary points at $x = \pm 1$, and 10,000 PDE collocation points randomly sampled inside the space–time domain. We use epoch-dependent loss weights to keep the PDE residual, initial condition, and boundary values balanced during training. This helps the network avoid situations where one part of the loss overwhelms the others, a common issue for Burgers' equation.

The model is trained first with the Adam optimizer (15,000 epochs with learning-rate decay). In many PINN studies, a second stage using L-BFGS sharpens the final solution, but for our experiments, the Adam stage already yielded a stable and accurate model.



Reference Comparison. To make the evaluation more reliable, we also compare the PINN prediction with a finite-difference reference solution generated using central spatial differences and an SSP–RK3 (Shu–Osher) time integrator. This numerical solution serves as a ground truth and helps confirm that the PINN captures the correct wave steepening and diffusion behavior, especially at later times.

Altogether, the Burgers equation provides a clear and well-understood setting to validate our PINN implementation before moving on to more complex nonlinear systems such as the Riemann problem.

4.2 Riemann Problem

The Riemann problem describes the evolution of an initial discontinuity under a system of hyperbolic conservation laws. In our case, the governing equations are one-dimensional gas-dynamic equations,

$$\rho_t + (\rho u)_x = 0, \tag{12}$$

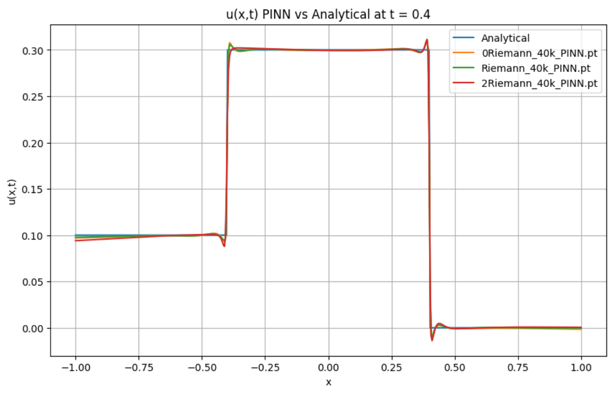
$$(\rho u)_t + (\rho u^2 + p)_x = 0, \tag{13}$$

which generate left- and right-moving waves and can develop steep gradients or shock-like features depending on the initial jump. A PINN must both represent sharp transitions and maintain stable gradients during training, two tasks that heavily depend on the choice of activation function.

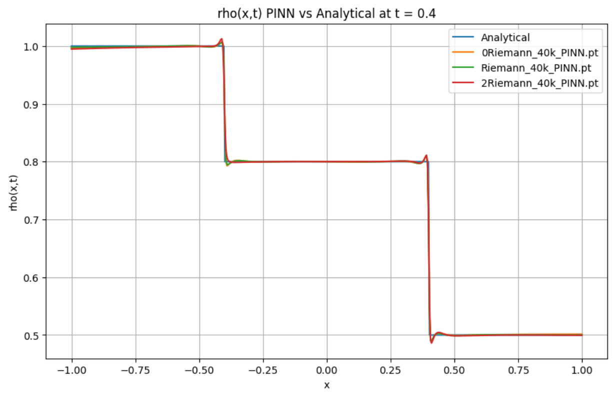
Motivation. During the project, the Riemann problem evolved into a natural playground for testing how activation functions affect PINN stability. While classical PINN papers mostly use `tanh`, the behavior of the Riemann solution suggested that smoother activations might smear out discontinuities, whereas piecewise-linear activations might help represent sharp jumps but introduce optimization difficulties. To explore this systematically, I trained multiple models using different activations and tracked how the loss curves and the predicted solutions behaved over long training runs.

Experiments with Tanh. The first set of experiments used the traditional `tanh` activation function on the domain $x \in [-1, 1]$, $t \in [0, 1]$.

- **40k epochs, no loss weights**: The longer training improved smooth regions but did not significantly improve the behavior around discontinuities. In the interior domain the results look like a Fourier reconstruction of the analytical solution. The loss curve remained stable, but the solution remained visibly incorrect in the shock region.
- **40k epochs, four-stage loss weighting**: Introducing progressively adjusted loss weights helped the network focus on the initial discontinuity, leading to smoother and more consistent solutions. This model performed noticeably better.
- **40k epochs, gradually changing loss weights**: Smoothly varying the weights during training produced more stable optimization and slightly improved the reconstruction of the left- and right-propagating waves.



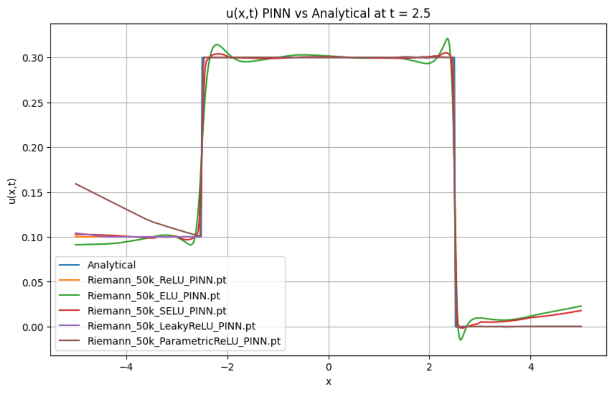
(a) $u(x, t)$ comparison



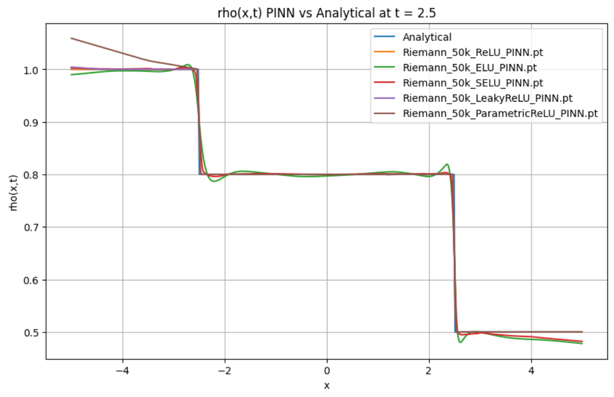
(b) $\rho(x, t)$ comparison

Experiments with ReLU-family activations. Because discontinuities are difficult for smooth activations, the next phase explored activations with piecewise-linear structure. These experiments used the larger domain $x \in [-5, 5]$, $t \in [0, 3]$, which produces more clearly separated waves.

- **ReLU (50k epochs)**: The loss plot was unstable, but interestingly, the predicted solution in the physical region was relatively accurate. This suggests that ReLU may help represent sharp features, but its non-smooth gradients make optimization noisy.
- **ELU (50k epochs)**: ELU produced a very stable loss curve, but the final solution did not match the analytical structure. The network became too smooth, failing to track the discontinuity.
- **SELU (50k epochs)**: Similar to ELU: stable optimization but unsatisfactory final results. The dissipative nature of SELU tended to blur the solution.
- **Leaky ReLU (50k epochs)**: This activation performed surprisingly well. The loss plot was noisy, yet the actual predicted solution aligned closely with the analytical Riemann structure. The small negative slope of Leaky ReLU seems to balance stability and expressiveness.
- **Parametric ReLU (50k epochs)**: The loss curve remained almost stable with only small fluctuations throughout training, suggesting that the optimization was not diverging. However, the final solution did not fully align with the analytical reference. The model captured the main shock structure fairly well in the interior of the domain, but it consistently failed to satisfy the boundary behaviour. This indicates that while PReLU can represent sharp features, the additional trainable parameter may interfere with the network's ability to enforce boundary conditions within the PINN framework.



(a) $u(x, t)$ comparison



(b) $\rho(x, t)$ comparison

Results and Discussion. Overall, the Riemann problem highlighted the sensitivity of PINNs to activation functions: smooth activations (`tanh`, `ELU`, `SELU`) train stably but tend to smooth out discontinuities, while piecewise-linear activations (ReLU family) can represent sharp jumps but often destabilize the optimization.

Among all tests, **Leaky ReLU** provided the best balance: the loss curve was irregular, but the predicted density and velocity fields matched the analytical Riemann solution more closely than any other activation.

These experiments demonstrated that for discontinuous hyperbolic systems, the activation function plays a role as important as the network architecture or sampling strategy. The experience also motivated the later implementation of more advanced techniques such as adaptive loss weighting and domain-specific sampling.

4.3 Three-Dimensional Heat Equation

To test whether the PINN framework could scale beyond one–dimensional PDEs, I implemented a model for the three–dimensional heat equation,

$$u_t = \alpha\,(u_{xx} + u_{yy} + u_{zz}), \tag{14}$$

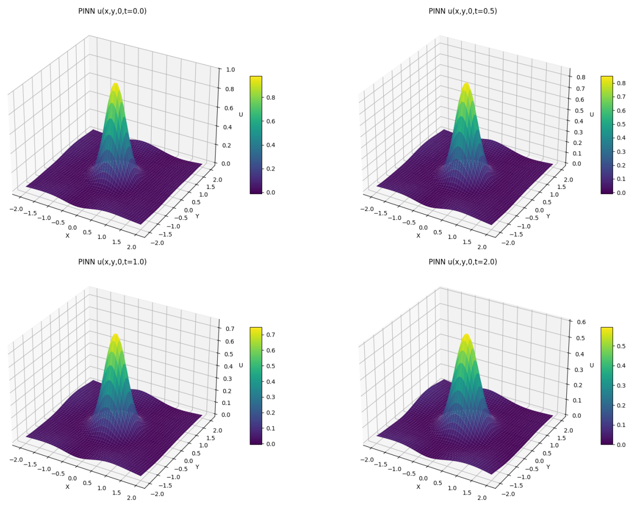
which requires the network to learn a smooth spatiotemporal diffusion pattern over four inputs: (x, y, z, t). This problem is significantly more demanding than the previous Burgers or Riemann setups because the PINN must approximate a mapping in $\mathbb{R}^4 \to \mathbb{R}$ and maintain stable derivatives in all directions. The experiment also provided an opportunity to test the effect of different hidden-layer sizes on the accuracy of the learned solution.

Activation and Training Setup. All experiments used a `tanh` activation function, which tends to work well for diffusive PDEs. The models were trained on the spatial domain $[-1, 1]^3$ and temporal interval $t \in [0, 1]$. A smooth 3D Gaussian,

$$u(x, y, z, 0) = \exp\bigl[-5(x^2 + y^2 + z^2)\bigr]\,,$$

was used as the initial condition. This setup also enabled comparison with the known analytical solution of the heat equation, which provided a reliable way to judge how accurately each network architecture generalized both inside and slightly outside the training domain.

Model 1: Deep Network with Large Width. The first experiment used the architecture

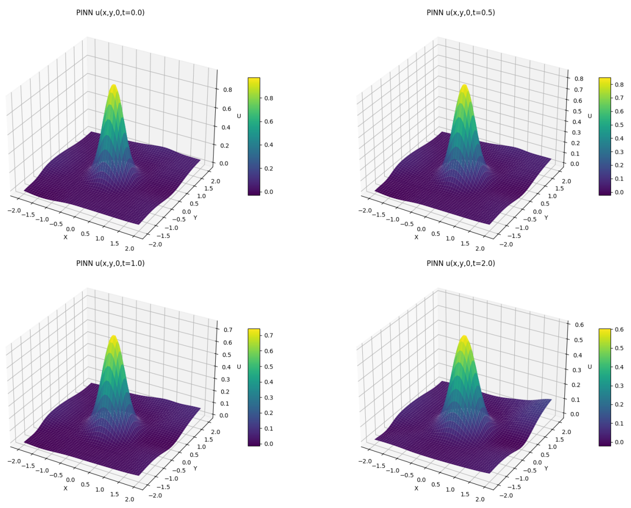


$$\texttt{layers} = [4,\ 64,\ 64,\ 64,\ 64,\ 1],$$

trained for 30,000 epochs. The loss became stable around 25,000 epochs, and the predicted solution matched the analytical Gaussian profile very well at $t = 1.5$ within the domain $[-1, 1]^3$. However, when evaluated outside this cube (for example $|x| > 1$, $|y| > 1$, or $|z| > 1$), the predictions began to deviate noticeably. This behavior was expected, PINNs generally interpolate well but extrapolate poorly, especially in high-dimensional settings where the PDE residual is enforced only within a finite region.

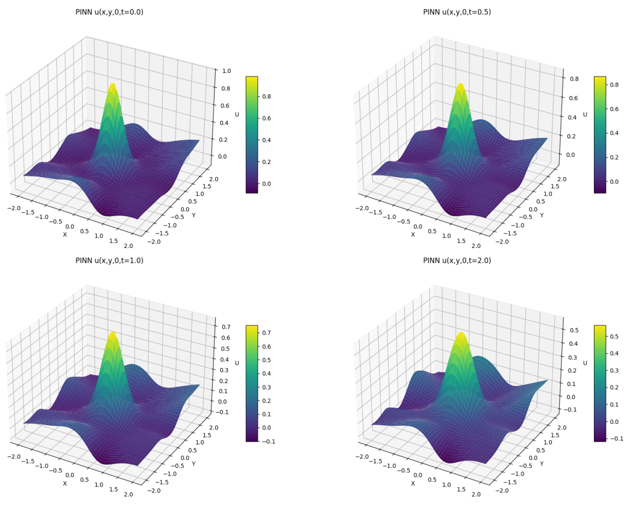
Model 2: Medium-Width Network. The second experiment reduced the hidden-layer sizes to

$$\texttt{layers} = [4,\ 32,\ 16,\ 8,\ 1].$$



Despite having far fewer parameters, the model trained very stably and produced results similar to the larger network inside the training cube. The loss curve remained smooth throughout all 30,000 epochs. This suggested that, for smooth diffusive problems like the heat equation, extremely wide layers may not be required to achieve accurate performance within the domain of interest.

Model 3: Shallow Network. A third experiment used a much smaller architecture,



$$\texttt{layers} = [4,\ 32,\ 8,\ 1],$$

again trained for 30,000 epochs. While the loss was still stable, the solution began to diverge more noticeably near the edges of the training domain and became inaccurate even for x, y, z slightly outside $[-1, 1]$. The reduced expressive capacity of the network limited its ability to represent the full shape of the diffusing Gaussian as time progressed. This model demonstrated clearly that network depth and representational power matter more in higher-dimensional PDEs than in one- or two-dimensional problems.

Results and Discussion. Across the three architectures, several patterns emerged:

- All models reproduced the analytical solution well inside the domain where they were trained.
- Larger or moderately sized networks handled temporal evolution better, especially at later times (e.g., $t = 1.5$).
- Extrapolation outside the training cube was consistently unreliable, with deeper networks performing better but still diverging beyond $|x|, |y|, |z| > 1$.

- The loss curves for all models were remarkably stable, confirming that the heat equation is a friendly benchmark for PINNs due to its smooth dynamics.

These results showed that PINNs can indeed handle four-dimensional input spaces, but architectural choices strongly influence the model's ability to generalize and maintain accuracy across the spatial domain. The 3D heat equation provided a useful contrast with the Riemann problem: whereas discontinuities challenge the activation function and training stability, the heat equation tests the network's depth and expressive capacity in high-dimensional smooth settings.

4.4 Kinematic Dynamo Model

To test the capability of PINNs on vector-valued, three-dimensional PDEs, I implemented the magnetic induction equation,

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{u} \times \mathbf{B}) + \frac{1}{Re_m} \nabla^2 \mathbf{B}, \qquad \nabla \cdot \mathbf{B} = 0, \tag{15}$$

which describes the evolution of a magnetic field $\mathbf{B}(x, y, z, t)$ in a prescribed velocity field. This problem is significantly more complex than the scalar PDEs attempted earlier, because the PINN must simultaneously predict three fields (B_x, B_y, B_z), and satisfy a full curl-based operator using automatic differentiation in four variables. The kinematic dynamo model is well known for producing exponential magnetic energy growth at sufficiently large magnetic Reynolds numbers Re_m, and I attempted to reproduce the dynamo behavior reported in Archontis et al. (2003).

Setup and ABC Flow. The velocity field used in the simulations is the Arnold–Beltrami–Childress (ABC) flow, a standard benchmark for kinematic dynamos. The spatial domain was the periodic cube $[0, 2\pi]^3$ and the temporal horizon was $t \in [0, 40]$. The PINN took (x, y, z, t) as input and returned (B_x, B_y, B_z). The loss function combined three terms:

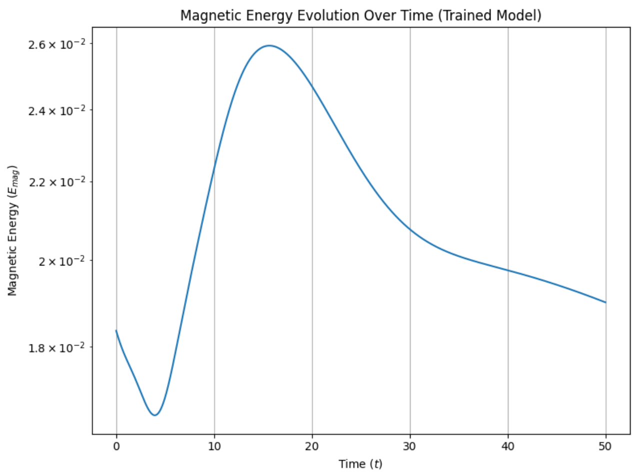
- the full vector PDE residual,
- the divergence-free condition $\nabla \cdot \mathbf{B} = 0$,
- and the initial seed field condition.

Several weighting strategies were tested, including square residuals, square-root residuals, and different coefficients for PDE, divergence, and initial-condition terms.

Experiments and Observations. Multiple models were trained under different magnetic Reynolds numbers, loss definitions, and PDE scalings:

- **Model 1: 10k epochs,** $Re_m = 100$

  File: `kinematic_dynamo_action_10k_tanh_NN_10lossicW_Rem100.pt`

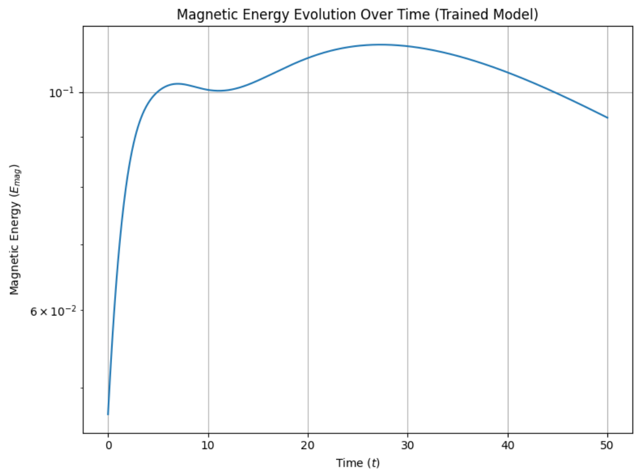


$$\text{loss}_{\text{pde}} = \left\langle \sqrt{f_{Bx}^2 + f_{By}^2 + f_{Bz}^2} \right\rangle ,$$

The loss dropped steadily but the magnetic energy curve did not show any clear exponential growth.

- **Model 2: 20k epochs,** $Re_m = 1600$

  File: `kinematic_dynamo_action_20k_tanh_NN_10lossicW.pt`

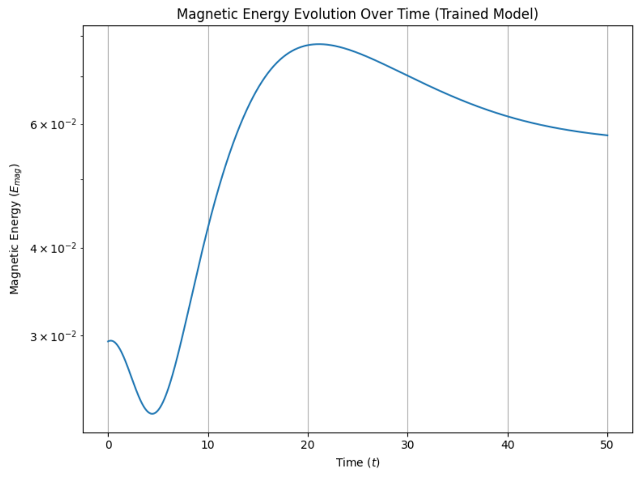


$$\text{loss}_{\text{pde}} = \left\langle f_{Bx}^2 + f_{By}^2 + f_{Bz}^2 \right\rangle ,$$

  The extended training phase did make the solutions smoother, but the magnetic energy still didn't match the behavior expected from a true dynamo.

- **Model 3: Modified PDE scaling,** $K = 3$**,** $Re_m = 100$

  File: `kinematic_dynamo_action_0k_tanh_NN_10lossicW_K=3.pt` Changing the PDE loss term to square-root PDE loss,



$$\text{loss}_{\text{pde}} = \left\langle \sqrt{f_{Bx}^2 + f_{By}^2 + f_{Bz}^2} \right\rangle ,$$

  and manually tuned weights,

$$\text{loss} = 10\,\text{loss}_{\text{pde}} + 10\,\text{loss}_{\text{div}} + \text{loss}_{\text{ic}},$$

  produced the most stable loss curve among the experiments. However, even in this case, the magnetic energy failed to exhibit the exponential growth phase characteristic of a working kinematic dynamo.

Comparison with Expected Dynamo Behaviour. The results from all PINN runs were compared with the growth curves reported in Archontis et al. (2003), but none of the trained models reproduced the true kinematic dynamo signature. The predicted magnetic energy:

- either decayed too rapidly,
- or plateaued at an arbitrary small value,

While the real dynamo system shows a clean exponential growth phase in magnetic energy, the PINN didn't capture any of that behavior. This gap makes it clear that the current network design isn't yet capable of handling such a complex, high-dimensional vector PDE, especially one where the different components are tightly coupled and strongly influence each other

Likely Reasons for Failure. The magnetic induction equation is the most complex PDE attempted in this project, combining:

- three coupled vector components,
- full three-dimensional Laplacians,
- curl operators involving all cross-derivatives,
- the divergence-free constraint,
- and exponential growth dynamics.

The simple fully connected network used in this work might be fine for scalar PDEs, but it's probably too limited to capture the complexity of the vector fields involved here.

Results and Discussion. Overall, the experiments indicate that a standard PINN with moderate depth and width is not sufficient to model 3D dynamo action. A much larger architecture, or a more advanced framework such as Fourier neural operators, physics-informed spectral networks, or domain decomposition methods, may be required. The kinematic dynamo therefore represents a demanding benchmark that exposes the limitations of classical PINNs when applied to multi-component, three-dimensional vector equations.

4.5 2D Kinematic Dynamo Action

In addition to the fully three-dimensional kinematic dynamo model, I also implemented a reduced two-dimensional variant of the induction equation. The purpose of this experiment was to check whether simplifying the input dimensionality helps the PINN to learn the magnetic field evolution more reliably. In this reduced system, spatial variation occurs only (y, z) while the velocity field contains a single nonzero component. The governing equation becomes

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{u} \times \mathbf{B}), \qquad \nabla \cdot \mathbf{B} = 0, \tag{16}$$

with velocity

$$\mathbf{u}(y, z) = \big(0,\, u_0 \sin(z/2),\, 0\big),$$

and initial magnetic field

$$\mathbf{B}(y, z, 0) = (0,\, 0,\, B_0).$$

The initial field points entirely in the z direction, and the velocity contains only a y component. Ideally, the network should preserve the initial field $t = 0$ and produce smooth growth or decay patterns at later times.

Model Variants. To test how architecture depth and width influence accuracy, several different PINN models were trained:

- **10k epochs, Tanh activation, 4 hidden layers $\times$ 64 neurons**

  (`2D_kinematic_dynamo_action_10k_tanh_x,y,z.pt`)

- **10k epochs, ReLU activation, 4 layers $\times$ 64**

  (`2D_kinematic_dynamo_action_10k.pt`)

- **10k epochs, ReLU, 4 layers $\times$ 128 neurons**

  (`2D_kinematic_dynamo_action_10k_128*4-hiddenlayers.pt`)

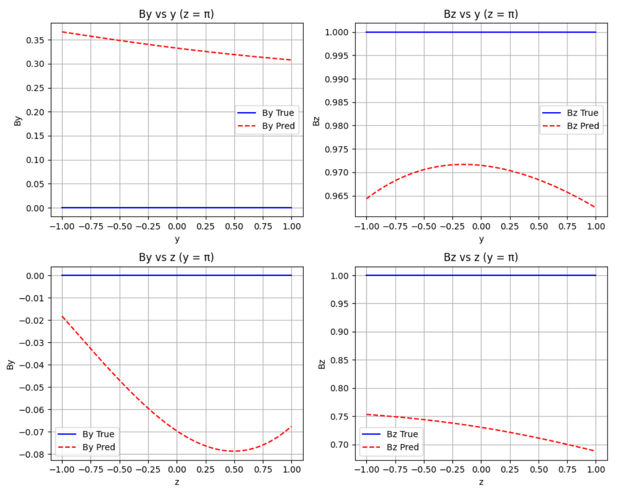
- **10k epochs, ReLU, 2 hidden layers $\times$ 64**

  (`2D_kinematic_dynamo_action_10k_64*2-hiddenlayers.pt`)

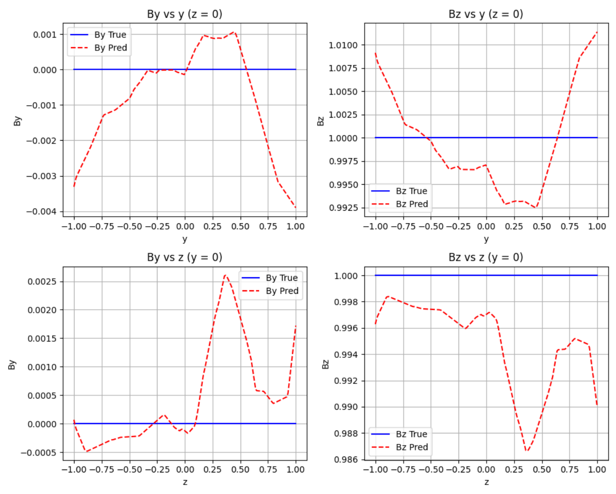
- **20k epochs, ReLU + cosine velocity profile, large decreasing architecture**

  $$\texttt{layers} = [3,\ 128,\ 64,\ 32,\ 16,\ 8,\ 4,\ 2]$$

  (`2D_kinematic_dynamo_action_20k_[3,128,64,32,16,8,4,2]-cos-Relu.pt`)



(a) B t = 0 comparison



(b) B t = 0 comparison

Results and Discussion. Across all the tested architectures, the loss curves were stable and decreased smoothly with training—yet the predicted magnetic field exhibited systematic errors.

- **Initial condition not satisfied.** Even at $t = 0$, the predicted field did not match $(0, 0, B_0)$ everywhere in the domain. The 2D slices shown in Fig. 4a and Fig. 4b illustrate that both B_y B_z contain spatial variations that should not exist. This problem persisted across all activations and layer widths.

- **Mismatch between 2D-input and 3D-input formulations.** The same velocity field and initial condition were implemented in both the full (x, y, z) PINN and the reduced (y, z) PINN. However, the two models produced *qualitatively different predictions*: - the 3D version produced a smooth surface (Fig. 4b), - the 2D version produced sharply bent or oscillatory profiles in B_y and B_z(Fig. 4a), especially visible in the B_y vs. z and B_z vs. $z$plots.

  This indicates that the dimensional reduction does not simply restrict the 3D model; rather, the optimization landscape changes substantially.

- **Field evolution inaccurate at later times.** At $t = 5$, the two–dimensional PINN produced a magnetic–magnitude surface that differed significantly from the behaviour expected under passive stretching. The predicted field lacked the correct smooth symmetry in z and showed no sign of exponential growth or clear decay.

Interpretation. Although the 2D kinematic dynamo case is simpler than the full 3D model, the PINN still struggled to learn the induction equation accurately. The core issues—failure to match the initial condition, inconsistencies between 2D and 3D inputs, and incorrect field evolution—suggest that the current PINN architectures are not expressive enough, or that the loss landscape is too stiff to be optimized effectively.

Given that even this reduced system failed to converge to physically meaningful solutions, it is likely that:

- a significantly larger model,

- more expressive activation functions (e.g. GeLU, Swish),
- Fourier neural operators or spectral PINNs,
- or a hybrid finite-difference + PINN formulation

would be needed to reproduce true kinematic dynamo behaviour in two or three dimensions.

Summary. The 2D experiments show that reducing dimensionality alone does not resolve the difficulties faced by PINNs when approximating vector–valued induction equations. The mismatch with analytical initial conditions and the inconsistency between 2D and 3D formulations highlight an important limitation: standard fully connected PINNs struggle with curl-based PDEs and magnetic-field constraints, even when the geometry is simplified.

5 Observations and Challenges

- **Smooth PDEs are well suited for PINNs.** Burgers and the 3D heat equation showed excellent agreement with analytical solutions when the solution remained smooth and the PDE was dominated by diffusion.
- **Discontinuities remain a core difficulty.** The Riemann problem demonstrated that even long training and activation tuning cannot fully overcome the shock-smoothing effect inherent to PINNs.
- **Higher-dimensional inputs require larger models.** When the input space expanded to (x, y, z, t), smaller architectures failed to capture the correct temporal evolution, and extrapolation outside the domain degraded rapidly.
- **Vector PDEs are significantly harder.** The kinematic dynamo experiments showed persistent difficulties enforcing $\nabla \cdot \mathbf{B} = 0$ and capturing curl-based coupling terms. Neither the 3D nor the 2D formulation produced physically meaningful evolution.
- **Training stability depends strongly on loss weighting and scheduling.** Without careful tuning of learning-rate decay and PDE/IC/divergence-balanced losses, the models often stalled or produced oscillatory residuals.

6 Conclusion

This project explored how well Physics-Informed Neural Networks can learn a wide range of PDEs, starting with smooth scalar systems and gradually moving toward more demanding, higher-dimensional vector problems. The results make a pattern very clear. When the physics is gentle and the solution behaves nicely, as in the Burgers equation or the 3D heat equation, the PINN settles into the solution quickly and produces accurate, visually clean predictions. Even the Riemann problem, despite its sharp discontinuity, revealed useful insights into how different activations, sampling strategies, and loss-balancing tricks influence convergence.

The picture changes when we step into the world of magnetic dynamos. Here the solution space becomes more tangled: strong coupling between vector components, exponential field growth, and three-dimensional structure all push the standard fully connected PINN well past its comfort zone. The models were able to learn fragments of the behavior, like partial field structures or smooth decay—but they consistently fell short of reproducing the expected exponential amplification or maintaining accurate initial conditions. These failures are not wasted effort, they underline a deeper point about the limitations of basic PINN architectures when the physics becomes high-dimensional, stiff, or strongly nonlinear.

Overall, the work shows both the promise and the boundaries of classical PINNs. They remain powerful tools for smooth, moderately complex PDEs, but scaling them to realistic vector-valued systems will require more sophisticated ideas, architectures that encode physical structure, adaptive sampling, Fourier-based representations, or hybrid neural-numerical methods. Exploring these directions would be a natural continuation of this project and a necessary step toward making PINNs practical for challenging problems like kinematic dynamos.

Code Availability

All code used in this project, including PINN implementations, training scripts, and visualization tools, is available at

https://github.com/Jaidhar-689/Physics-Informed-Neural-Networks-for-Nonlinear-PDEs

References

[1] M. Raissi, P. Perdikaris, and G. 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, vol. 378, pp. 686–707, 2019.

[2] K. Sun, Z. Zhang, and Q. Zeng, *Applications of Physics-Informed Neural Networks in Computational Physics*, Computer Methods in Applied Mechanics and Engineering, 2022.

[3] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis, *Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks*, Proceedings of the Royal Society A, vol. 476, 2020.

[4] N. Martinez, K. Pawar, and A. Stuart, *Training Neural Networks in Sobolev Spaces*, SIAM Journal on Applied Mathematics, 2022.

[5] Y. Wang, H. Wang, and J. Perdikaris, *On the curse of domain scaling in physics-informed neural networks*, SIAM Journal on Scientific Computing, 2022.