

# Physics-Informed Neural Networks for Computational Fluid Dynamics Validation Using DeepXDE

Sai Krishna K<sup>1</sup>, Bibek Dhungana<sup>2</sup>, and Chandan Bose<sup>3</sup>

<sup>1</sup>UG Student, Department of Mechanical Engineering, Thapar Institute of Engineering and Technology

<sup>2</sup>Assistant Professor, Department of Mechanical Engineering, Tribhuvan University

<sup>3</sup>Assistant Professor, Aerospace Engineering, College of Engineering and Physical Sciences, The University of Birmingham

## Abstract

PINNs have recently gained traction as an innovative method for addressing and verifying issues tied to partial differential equations, presenting a fresh perspective compared to conventional CFD techniques. This study investigates the application of PINNs, implemented via the DeepXDE framework [2], to simulate and validate two-dimensional laminar lid-driven cavity flow, a well-established CFD benchmark. By embedding the Navier–Stokes equations into the neural network’s loss function, PINNs enforce physical consistency while incorporating boundary conditions to capture flow dynamics accurately. Confirming the accuracy of CFD simulations with traditional tools like OpenFOAM demands significant computational effort, involving detailed meshing and discretization processes. In contrast, PINNs reduce reliance on these processes, potentially enhancing computational efficiency. The PINN model’s results are compared with OpenFOAM simulations, demonstrating its ability to approximate velocity fields and flow patterns with reasonable accuracy. Additionally, this study explores the scalability of PINNs for more complex flow problems and their potential integration with high-performance computing environments. This work highlights PINNs as a complementary tool for CFD validation, offering insights into their efficiency and adaptability for laminar flow problems.

## 1 Introduction

Over the past several decades, computational fluid dynamics (CFD) has made incredible progress, enabling engineers and scientists to simulate a wide range of phenomena, from the gentle flow of air in a room to the aerodynamic forces on a racing car. Traditional methods—like finite element, spectral, and mesh-free techniques—have become the backbone of how we approach these complex simulations. Yet, despite these advances, specific core challenges remain. For instance, creating high-quality computational meshes is still something of an art form and can be painstakingly time-

consuming, especially for industrial applications. Integrating experimental data of varying fidelity into simulations is rarely straightforward, and tackling inverse problems—such as inferring unknown boundary conditions or properties—often demands entirely new algorithms and significant computational resources. Modern CFD software, such as OpenFOAM, has evolved into sprawling codebases that are challenging to maintain and update in line with the advancement of evolving technologies [3].

Over the past few years, deep learning has significantly impacted computational mechanics, excelling at representing complex, nonlinear systems that vary over space and time. Initial applications of deep learning in this area were predominantly data-driven, relying on neural networks as opaque models to connect inputs and outputs using extensive, well-prepared datasets. However, gathering enough high-fidelity data for training remains expensive and time-consuming, limiting the effectiveness of purely data-driven strategies [4].

To overcome these limitations, the field has begun to embrace physics-informed deep learning. Rather than ignoring the physical rules that govern nature, this approach weaves them directly into the neural networks. By embedding the governing equations—like the Navier–Stokes equations that describe fluid flow—along with initial and boundary conditions, these models become far more robust and interpretable. They require fewer data and are more effective in producing solutions that comply with physical laws [1].

A leading example of this new approach is the Physics-Informed Neural Network, or PINN. Pioneered in recent years, PINNs take the powerful machinery of neural networks and “teach” them physics by crafting a loss function that penalizes violations of known scientific principles. Thanks to tools like automatic differentiation, PINNs can enforce differential equations without the need for explicit mesh generation, bypassing one of the major pain points of conventional CFD. Moreover, the same PINN framework handles both traditional forward simulations and challenging inverse problems alike. While today’s PINNs might not yet match the raw speed and pinpoint accuracy of mature CFD solvers on standard benchmark problems, they really shine when experimental or simulation data is scarce or scattered, such as in real-world engineering or biomedical flows. PINNs also provide a natural way to combine available measurements with physical models for improved reliability [5].

Recent advances have seen PINNs successfully tackle problems ranging from the classic lid-driven cavity flow to more complicated flows around cylinders and even flows in biomedical contexts. Their ability to handle both data and physics makes them particularly well-suited for CFD validation, where new modelling techniques must be assessed against trusted simulation tools and experimental results [6].

In this report, we delve into the emerging field of physics-informed deep learning for CFD validation. We’ll compare the effectiveness of PINNs, using the DeepXDE framework, against the widely trusted OpenFOAM solver on the classical lid-driven cavity problem. By examining both the underlying numerical strategies and the results, we aim to highlight not only where PINNs are strong, but also where traditional methods still have an edge, and to explore what the future may hold as these technologies continue to mature.

## 2 Computational Methodology

The Navier-Stokes equations control the behavior of incompressible fluids and serve as a core principle in CFD.. These equations are given by:

$$\nabla \cdot \mathbf{u} = 0, \quad (1)$$

$$\rho \left( \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla) \mathbf{u} \right) = -\nabla p + \mu \nabla^2 \mathbf{u} + \mathbf{f}, \quad (2)$$

where  $\mathbf{u}$  is the velocity vector,  $\rho$  is the fluid density,  $p$  is the pressure,  $\mu$  is the dynamic viscosity, and  $\mathbf{f}$  represents body forces. Equation (1), known as the continuity equation, ensures the conservation of mass by stating that the divergence of the velocity field  $\mathbf{u}$  is zero, reflecting the incompressibility of the fluid. This implies that the volume of fluid is conserved as it moves through the domain. Equation (2), the momentum equation, describes the conservation of momentum and balances the inertial forces (left-hand side) with pressure gradients, viscous forces, and external body forces (right-hand side).

Expanded in Cartesian coordinates (x, y, z), the continuity equation becomes:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0, \quad (3)$$

where  $u$ ,  $v$ , and  $w$  are the velocity components in the x, y, and z directions, respectively.

The momentum equations in each direction are:

For the x-direction:

$$\rho \left( \frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} + w \frac{\partial u}{\partial z} \right) = -\frac{\partial p}{\partial x} + \mu \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) + f_x, \quad (4)$$

For the y-direction:

$$\rho \left( \frac{\partial v}{\partial t} + u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} + w \frac{\partial v}{\partial z} \right) = -\frac{\partial p}{\partial y} + \mu \left( \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2} \right) + f_y, \quad (5)$$

For the z-direction:

$$\rho \left( \frac{\partial w}{\partial t} + u \frac{\partial w}{\partial x} + v \frac{\partial w}{\partial y} + w \frac{\partial w}{\partial z} \right) = -\frac{\partial p}{\partial z} + \mu \left( \frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2} \right) + f_z. \quad (6)$$

The variables in these equations are defined as follows:

- $\mathbf{u} = (u, v, w)$ : The velocity vector, a three-dimensional field representing the speed and direction of fluid flow at any point in the domain. It is a critical output in CFD simulations.
- $\rho$ : The fluid density, a scalar quantity that measures the mass per unit volume of the fluid, is assumed constant for incompressible flows.

- $p$ : Pressure ( $p$ ): This scalar field reflects the fluid's normal force per area, directly affecting the flow's magnitude and path.
- $\mu$ : The dynamic viscosity, a measure of a fluid's resistance to shear or angular deformation, which governs the viscous forces in the flow.
- $\mathbf{f} = (f_x, f_y, f_z)$ : The body force vector, representing external forces acting on the fluid, such as gravity or electromagnetic forces, which can influence the overall motion.

This study utilises Physics-Informed Neural Networks (PINNs) to solve these equations for CFD validation, embedding them directly into the neural network's loss function to enforce physical constraints during training.

## 2.1 Problem Description

The lid-driven cavity flow problem is defined in a unit square domain ( $[0,1] \times [0,1]$ ) with no-slip boundary conditions on the bottom, left, and right walls ( $u = v = 0$ ) and a constant velocity ( $u = 1, v = 0$ ) on the top wall. The flow is modelled as incompressible and laminar with a Reynolds number of  $Re = 40, 100$  where  $Re = UL/\nu$ ,  $U = 1$  (lid velocity),  $L = 1$  (cavity length), and  $\nu$  is the kinematic viscosity. The governing equations are the steady-state incompressible Navier-Stokes equations, with  $\rho = 1$  and  $\nu = 1/40, 1/100$ .

## 2.2 PINN Implementation

The PINN model was constructed using the DeepXDE framework[2], which is a dedicated Python library for resolving differential equations via neural networks. The neural network's loss function incorporates the Navier-Stokes equations by summing the residuals from the continuity equation and the two momentum components ( $x$  and  $y$ ). The loss function is weighted with  $[1, 1, 3]$  for the PDE residuals and  $[2]$  for each boundary condition to balance their contributions. The chosen architecture is a feedforward neural network with 2 inputs ( $x, y$ ) and 3 outputs ( $u, v, p$ ). It utilizes 6 hidden layers, each with 50 neurons, employing the tanh activation and Glorot uniform initialization. The geometry is defined as a unit square using `dde.geometry.Rectangle([0, 0], [1, 1])`. Boundary conditions are enforced using Dirichlet boundary conditions:  $u = 1, v = 0$  on the top wall ( $y = 1$ ), and  $u = v = 0$  on the other walls. To eliminate pressure indeterminacy, the pressure is fixed at  $p = 0$  at the point  $(0.5, 0.5)$ . Training was executed with a dataset of 20,000 internal collocation points and 4,000 boundary points (uniformly sampled, 1,000 per wall), anchored by a fixed pressure reference point. The model is trained using the Adam optimiser with a learning rate of  $1e-3$  for 20,000 iterations, with model checkpoints saved every 1,000 iterations. The final model is saved as `model-lid-re40.pt`.

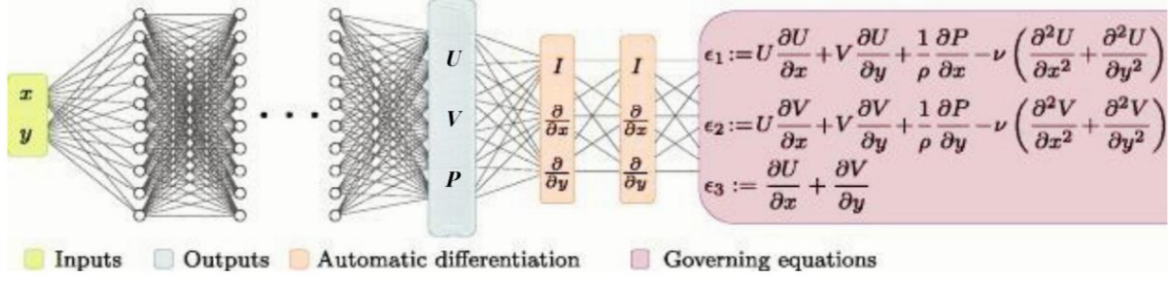


Figure 1: Schematic representation of a Physics-Informed Neural Network (PINN) for solving incompressible Navier-Stokes equations. The network takes spatial inputs ( $x, y$ ), predicts velocity and pressure, and computes PDE residuals via automatic differentiation.

The total loss function in PINNs is a weighted combination of two terms: the boundary condition loss and the PDE residual loss.

$$\mathbf{U}_b^n(\theta) = [u, v, p]_{\text{pred}}, \quad (7)$$

$$\tilde{\mathbf{U}}_b^n = [u, v, p]_{\text{true}}, \quad (8)$$

where  $\theta$  represents the network parameters. The boundary loss  $L_b$  is defined as:

$$L_b = \frac{1}{N_b} \sum_{n=1}^{N_b} |\mathbf{U}_b^n(\theta) - \tilde{\mathbf{U}}_b^n|^2 \quad (9)$$

The residual loss  $L_r$  is computed over the PDE residuals as:

$$L_r = \frac{1}{N_r} \sum_{i=1}^3 \sum_{j=1}^{N_r} w_i \epsilon_{i,j}^2 \quad (10)$$

where  $N_b$  and  $N_r$  denote the number of boundary and residual points respectively,  $w_i$  represents the weighting for each equation residual  $\epsilon_i$ . The total loss is then given by:

$$L = L_r + \beta L_b \quad (11)$$

## 2.3 DeepXDE Framework

DeepXDE (Deep Learning for Differential Equations) is an open-source Python library for solving differential equations using Physics-Informed Neural Networks (PINNs). It provides a modular design that separates physical laws, computational domain, boundary conditions, and neural network training into distinct components. This makes it especially suitable for scientific machine learning tasks such as fluid flow modelling.

In this work, DeepXDE is employed to solve the two-dimensional incompressible Navier–Stokes equations for the classical lid-driven cavity problem at various Reynolds numbers. The trained PINN predicts velocity and pressure fields inside the cavity, which are later compared against OpenFOAM results for validation.

## Key Features of DeepXDE

- Automatic differentiation for evaluating PDE residuals without manual derivation.
- Flexible handling of geometries, sampling strategies, and collocation points.
- Direct support for boundary and initial conditions.
- Neural network backends compatible with both TensorFlow and PyTorch.
- Training utilities such as checkpointing and loss weighting.

## Program Workflow

The overall workflow of the PINN implementation can be summarised as follows:

1. **Define PDEs:** The Navier–Stokes momentum and continuity equations are written in terms of velocity components  $u, v$  and pressure  $p$ . Derivatives are computed automatically using DeepXDE’s gradient operators.
2. **Specify Geometry:** A square domain  $\Omega = [0, 1] \times [0, 1]$  represents the cavity.
3. **Apply Boundary Conditions:**
  - Top lid:  $u = 1, v = 0$  (driven wall).
  - Other walls:  $u = v = 0$  (no-slip).
4. **Construct PDE Data:** Residuals, geometry, boundary conditions, and collocation points are assembled into a dataset for training.
5. **Design Neural Network:** A feedforward network with six hidden layers and 50 neurons per layer (activation: tanh) serves as the trial solution.
6. **Train Model:** The Adam optimizer was used to minimize the weighted loss (PDE residuals and boundary penalties) over 20,000 iterations, with checkpoints saved throughout.
7. **Output Results:** After convergence, velocity and pressure fields are exported for visualisation and validation.

## Workflow Illustration

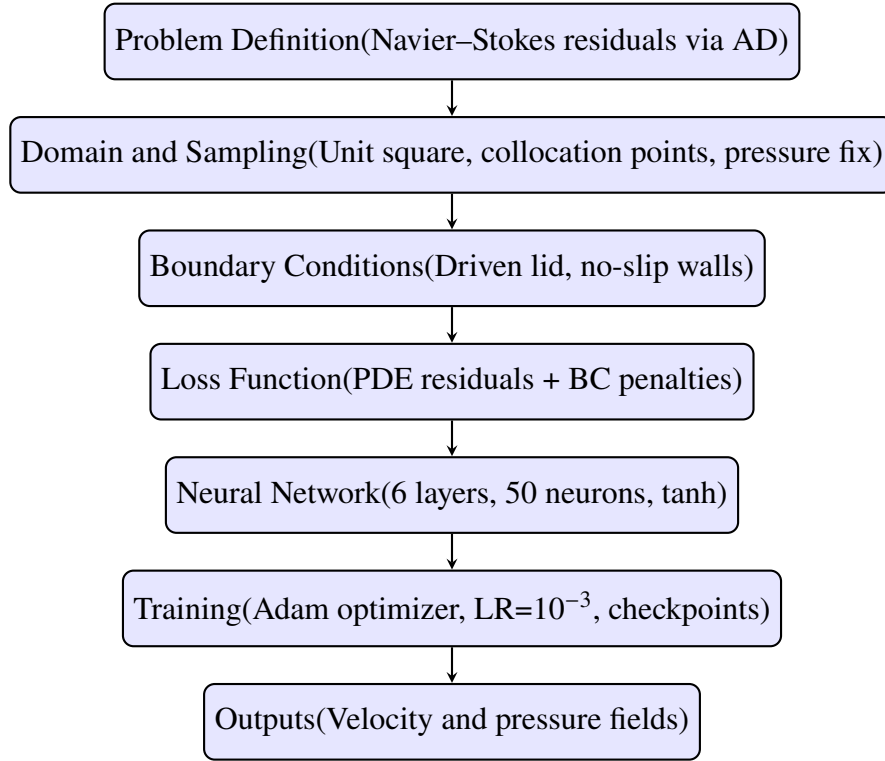


Figure 2: Workflow of the DeepXDE-based PINN implementation for lid-driven cavity flow.

## Explanation of Workflow Components

- **PDE Definition:** Encodes the momentum and continuity equations, with derivatives evaluated by automatic differentiation.
- **Geometry and Sampling:** The unit square cavity is discretised with collocation points in the interior and on boundaries. An anchor point fixes pressure.
- **Boundary Conditions:** The moving lid enforces flow, while the other walls enforce no-slip. Pressure fixation prevents singularity.
- **Loss Function:** Combines PDE residual errors with boundary condition mismatches, ensuring physical consistency.
- **Neural Network:** A fully connected feedforward network approximates  $(u, v, p)$  over the domain.
- **Training:** The optimiser reduces the total loss iteratively, with checkpoints for reproducibility.
- **Outputs:** After training, the model produces continuous approximations of velocity and pressure that can be directly compared to OpenFOAM.

## Pseudo Code

---

**Algorithm 1** Pseudo-code for PINN implementation using DeepXDE
 

---

- 1: Import necessary libraries (DeepXDE, TensorFlow, NumPy, etc.)
  - 2: Define Navier–Stokes PDE residuals and continuity equation
  - 3: Specify computational domain (unit square) and collocation points
  - 4: Apply boundary conditions:
    - No-slip on walls ( $u = v = 0$ )
    - Lid-driven boundary ( $u = 1, v = 0$ )
    - Pressure reference at a point
  - 5: Construct loss function = PDE residuals + BC penalties
  - 6: Define neural network architecture (6 hidden layers, 50 neurons each, tanh activation)
  - 7: Initialise optimisers:
    - Adam optimiser (20,000 iterations)
    - L-BFGS (for fine-tuning)
  - 8: Train network until convergence
  - 9: Save checkpoints and trained model
  - 10: Predict velocity and pressure fields on test grid
  - 11: Compare results with OpenFOAM reference data
- 

## Summary

DeepXDE provides a clear separation between physical laws, geometry, boundary enforcement, and the neural network ansatz. This modularity allows straightforward modifications such as changing Reynolds numbers, geometries, or network architectures. The resulting PINN solutions are well-suited for direct validation against CFD solvers such as OpenFOAM (see Section 2.4).

## 2.4 OpenFOAM Setup

For comparison with the PINN solution, OpenFOAM, —a widely utilized open-source finite volume CFD solver—was used to produce the reference results. To ensure a fair, direct comparison, the lid-driven cavity problem was run in OpenFOAM using the identical physical parameters and geometry as the DeepXDE model.

### Case Setup

- **Domain:** Square cavity  $\Omega = [0, 1] \times [0, 1]$ .
- **Boundary Conditions:**
  - Top lid:  $u = 1, v = 0$  (driven wall).
  - Remaining walls:  $u = v = 0$  (no-slip).
- **Reynolds Number:**  $Re = 40, 100$ .



### Solver and Mesh

- **Solver:** `icoFoam`, designed for incompressible, laminar, transient flow.
- **Mesh:** A Structured Cartesian grid with uniform cell spacing was generated, optimizing for a balance between solution accuracy and resource expenditure.
- **Time Advancement:** Transient simulation is run until a steady state is reached.

### Simulation Outputs

- Velocity components  $u(x, y)$  and  $v(x, y)$ .
- Pressure field  $p(x, y)$  (up to an arbitrary constant).
- Derived quantities such as vorticity for post-processing.

### Workflow Illustration

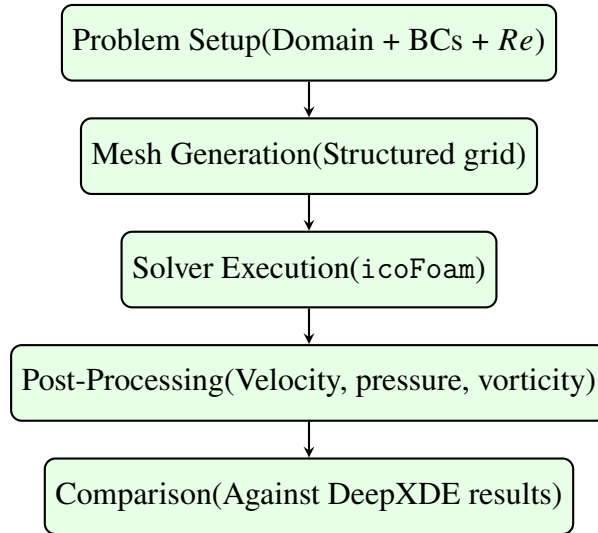


Figure 3: Workflow of OpenFOAM simulation for lid-driven cavity flow at  $Re = 40, 100$ .

### Summary

The OpenFOAM solution provides a high-fidelity CFD reference for the lid-driven cavity benchmark. By using identical geometry, Reynolds number, and boundary conditions as in the DeepXDE model, the comparison remains fair and consistent. The velocity and pressure fields obtained here are used in error evaluation between PINN results and OpenFoam to quantify the PINN accuracy and assess its ability to reproduce classical CFD results.

### 3 Results and Discussions

#### 3.1 PINN Results

The PINN model, trained for 20,000 iterations, successfully predicts the velocity and pressure fields for the laminar lid-driven cavity flow at  $Re = 40, 100$ . Velocity profiles along the vertical centerline ( $u$  at  $x = 0.5$ ) and horizontal centerline ( $v$  at  $y = 0.5$ ) exhibit the expected parabolic-like shapes characteristic of laminar cavity flow, with a primary vortex forming in the domain. The loss function converges steadily, indicating stable training, with the continuity equation weighted higher (weight = 3) to enforce incompressibility. Streamline plots generated from PINN predictions show a single dominant vortex, consistent with literature expectations for  $Re = 40, 100$ .

#### 3.2 OpenFOAM Results

OpenFOAM simulations using `icoFoam` produce high-fidelity velocity and pressure fields for the same conditions. The velocity profiles along  $x = 0.5$  and  $y = 0.5$  align closely with benchmark data from Ghia et al.[8], confirming the accuracy of the finite volume method. The mesh resolution ensures convergence, with negligible numerical artefacts in the streamline patterns, which also depict a single primary vortex.

#### 3.3 Comparison

Comparing PINN and OpenFOAM results, the velocity profiles along the centerlines show good agreement, with L2 norm errors on the order of  $10^{-2}$  for the  $u$  and  $v$  components relative to OpenFOAM. Discrepancies are primarily observed near the domain boundaries, where PINNs tend to slightly underpredict velocity gradients due to the soft enforcement of Dirichlet boundary conditions. Streamline visualisations from both methods reveal a single dominant vortex with comparable core positions. However, the PINN solution exhibits smoother transitions due to the continuous and differentiable nature of neural network representations.

In terms of computational performance, OpenFOAM simulations complete in under 10 minutes on a standard CPU with a  $100 \times 100$  mesh. The PINN model, implemented in Python using DeepXDE and trained for 20,000 iterations, required approximately 1.5 hours to run on a Dell Inspiron 5418 laptop equipped with an Intel Core i7-11390H CPU and an NVIDIA MX450 GPU. While PINNs involve longer training times, they offer significant advantages such as mesh-free modelling and seamless integration of physical laws, making them particularly attractive for complex geometries or inverse problems. The PINN training time could potentially be reduced by leveraging advanced GPU hardware or optimizing the neural network architecture, such as reducing the number of layers or using adaptive sampling strategies.

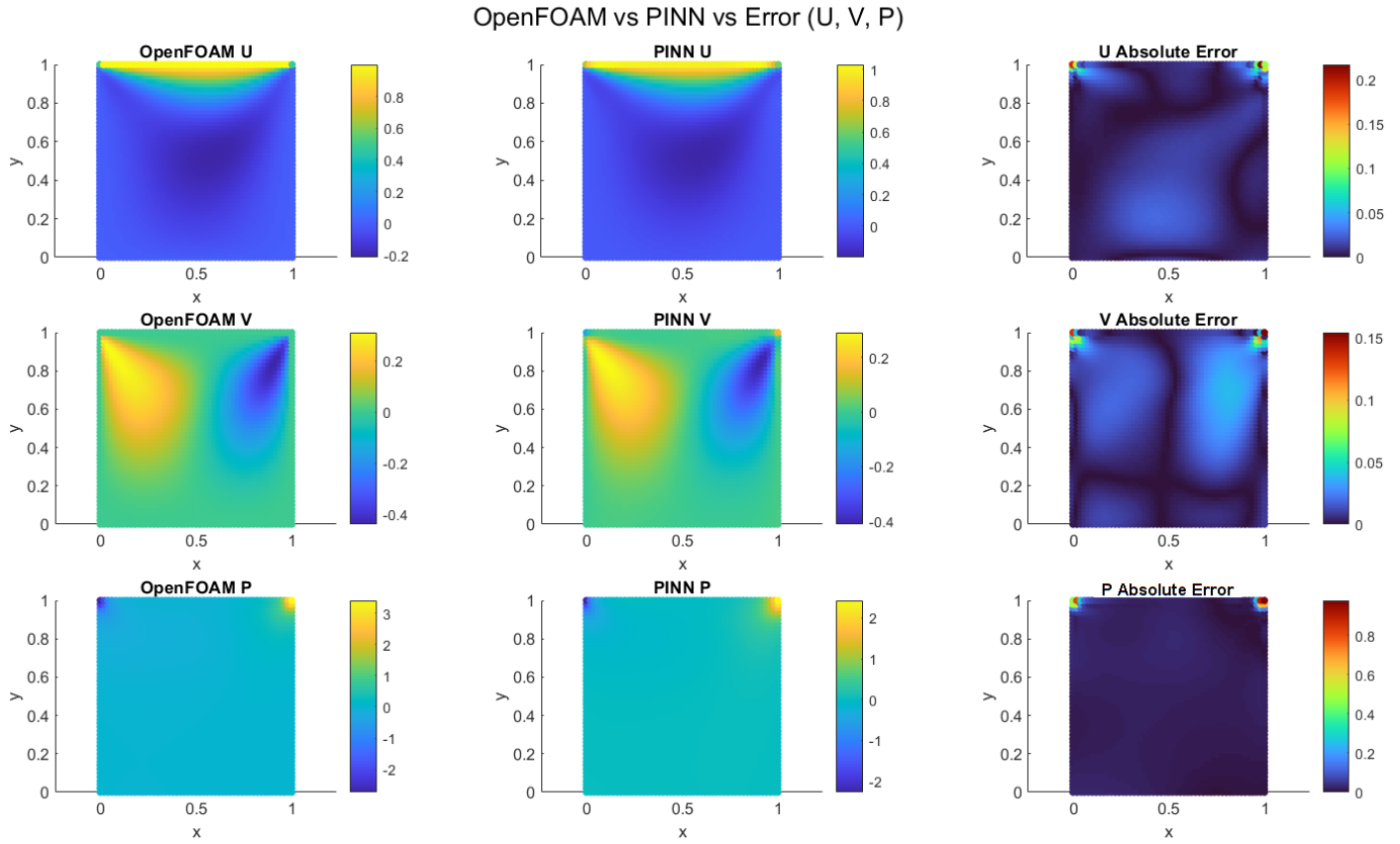


Figure 4: Comparison of velocity and pressure fields for lid-driven cavity flow at  $Re = 40$  between OpenFOAM and PINN predictions. Left column: OpenFOAM results; Centre column: PINN predictions; Right column: Absolute error maps.

The figures 4 and 5 above present a direct comparison between OpenFOAM and PINN results for the lid-driven cavity problem at Reynolds numbers  $Re = 40$  and  $Re = 100$ . Each row represents one component: the top row for horizontal velocity ( $u$ ), the middle for vertical velocity ( $v$ ), and the bottom for pressure ( $p$ ). The left column shows results from OpenFOAM, the middle column shows PINN predictions, and the right column visualises the absolute errors between the two.

The absolute error plots indicate that PINNs closely approximate OpenFOAM solutions across the domain, with notable discrepancies only near corners or boundary layers, where enforcing strict boundary conditions is more challenging for neural networks. The results highlight the effectiveness of PINNs in capturing key flow features without relying on traditional meshing or discretisation.

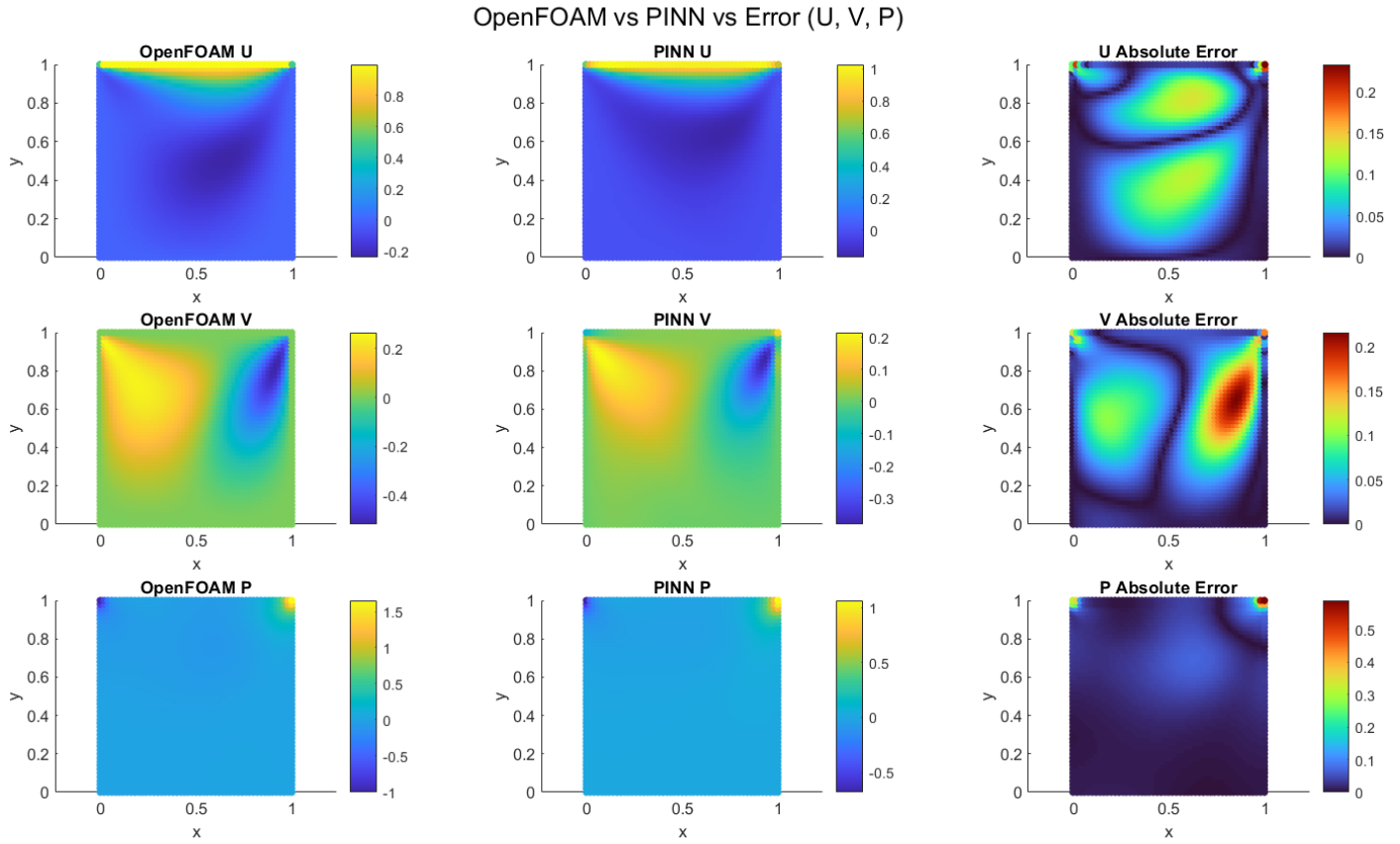


Figure 5: Comparison of velocity and pressure fields for lid-driven cavity flow at  $Re = 100$  between OpenFOAM and PINN predictions. Left column: OpenFOAM results; Centre column: PINN predictions; Right column: Absolute error maps.

## 4 Conclusions

This study demonstrates that Physics-Informed Neural Networks (PINNs), implemented using the DeepXDE framework, can accurately predict laminar lid-driven cavity flow by directly embedding the Navier-Stokes equations and relevant boundary conditions into the neural network's loss function. The PINN model achieved solutions for velocity and pressure fields that closely match the results from the conventional OpenFOAM CFD solver, with low L2 norm errors and reasonable agreement in the centerline velocity profiles and streamline patterns across the domain.

The mesh-free nature of the PINN methodology is its chief benefit, as it removes the laborious step of generating meshes, saving considerable time. Instead, the method operates on randomly sampled points within the computational domain and its boundary. Embedding physical laws directly into the network provides both physical consistency and flexibility, enabling the framework to tackle both standard forward and challenging inverse problems.

However, the PINN model requires greater computational resources in terms of training time com-

pared to OpenFOAM for this specific problem and Reynolds number. Successful training also depends on the appropriate selection of network architectures, loss weights, and optimisation strategies; improper choices can hinder convergence or accuracy, especially for more complex or higher Reynolds number problems.

Overall, the results affirm that PINNs are a valuable and complementary tool for CFD validation, particularly advantageous for mesh-free modeling, inverse problems, or scenarios where integrating scattered experimental or simulation data is needed. Although PINNs may not yet rival the efficiency of established CFD solvers for standard benchmark cases, they offer substantial promise for tackling challenging geometries, data-driven modeling, and future advances in scientific machine learning for fluid mechanics.

## 5 Future Work

While this research effectively used DeepXDE-based PINNs to simulate the lid-driven cavity flow and validate against OpenFOAM, the work can be expanded in several important, future-oriented directions.

- **Higher Reynolds Number Flows:** To model more challenging flows (like  $Re = 1000$ ) and capture their increased nonlinearity and vortical structures, PINNs will require advanced architectures, such as adaptive loss weighting or Fourier Neural Operators.
- **Three-Dimensional Simulations:** The present work is limited to two-dimensional geometry. Extending to three dimensions will allow more realistic flow scenarios and further evaluation of DeepXDE's scalability.
- **Unsteady Flow Problems:** Incorporating the time dimension into the PINN formulation to capture transient phenomena such as vortex shedding or flow development stages is a crucial next step. This would enable studying periodic and transient lid-driven cavity dynamics.
- **Hybrid Data Coupling:** Integrating PINNs with limited experimental or CFD measurements in a hybrid model is a strategy that could decrease training time and boost predictive accuracy, particularly in intricate geometries.
- **Turbulent Flow Modeling:** Investigating the applicability of PINNs to turbulence modeling, either through Reynolds-average Navier-Stokes (RANS) or Large Eddy Simulation (LES) formulations, is an important future step.
- **Improved Neural Network Architectures:** Exploring advanced architectures such as Fourier Neural Operators, attention-based models, or physics-informed transformers could enhance convergence and generalization capabilities.

In general, this work lays the foundation for physics-informed machine learning in fluid mechanics. Future developments will focus on extending the methodology to more complex, high-dimensional, unsteady, and industrially relevant flow problems.

## References

- [1] Raissi, M., Perdikaris, P., Karniadakis, G.E., “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.
- [2] Lu et al. (2021), DeepXDE: A deep learning library for solving differential equations, *SIAM Review*.
- [3] Ferziger, JH, Peri, M., *Computational Methods for Fluid Dynamics*. Springer, 3rd ed., 2002.
- [4] Brunton, S.L., Noack, B.R., Koumoutsakos, P., ‘Machine learning for fluid mechanics’, *Annual Review of Fluid Mechanics*, vol. 52, pp. 477–508, 2020.
- [5] Botarelli et al., “A review of physics-informed neural networks for CFD validation”, *AIP Advances*, vol. 14(3), 2024.
- [6] Karniadakis, G.E., et al., ‘Physics-informed machine learning’, *Nature Reviews Physics*, vol. 3, pp. 422–440, 2021.
- [7] U. Ghia, K. N. Ghia, and C. T. Shin, “High-Re solutions for incompressible flow using the Navier–Stokes equations and a multigrid method,” *Journal of Computational Physics*, vol. 48, no. 3, pp. 387–411, 1982.