KFUPM
Journal of Undergraduate Research International

Diamond Open Access 57

Article Computer Science/Mathematics

A Study of Physics-Informed Neural Networks for Solving Ordinary and Partial Differential Equations

Osama Hawladar and Abdullah Shah*

Cite https://doi.org/10.64589/juri/231590

Submitted: January 23, 2026 **Revised:** May 05, 2026 **Accepted:** May 16, 2026

ABSTRACT

Physics-informed neural networks (PINNs) have recently emerged as promising alternatives to traditional numerical methods for solving differential equations by directly incorporating governing physical laws into the learning process. However, standard PINNs (std-PINNs) often have difficulty handling nonlinear and stiff partial differential equations (PDEs), particularly in terms of stability and convergence. In this study, we investigate the application of PINNs to solve ordinary differential equations (ODEs) and PDEs, including the damped harmonic oscillator (an ODE), 1D heat equation (a linear PDE), and 1D and 2D Allen–Cahn equations (non-linear PDEs). Std-PINNs are first applied to linear problems to evaluate their baseline performance. To address the limitations of std-PINNs in nonlinear settings, a backward-compatible PINN (bc-PINN) framework is implemented, which decomposes the temporal domain into smaller segments and enforces solution continuity across these segments. The results reveal that std-PINNs achieve high accuracy for simple linear problems but fail to capture the dynamics of nonlinear equations, such as the Allen–Cahn equation. By contrast, bc-PINNs significantly improve prediction accuracy and stability on nonlinear and time-dependent PDEs, successfully approximating complex solution behaviors. These findings demonstrate the effectiveness of bc-PINNs as a robust extension of std-PINNs for solving challenging differential equations.

Keywords: physics-informed neural networks; differential equations; Allen–Cahn equation; heat equation; deep learning; computational

1. INTRODUCTION

Differential equations form the mathematical backbone of numerous scientific and engineering disciplines, encapsulating the laws governing dynamic systems across time and space. These equations describe a wide range of physical phenomena, including particle motion [1], heat diffusion [2], fluid flows [3], and structural deformations [4]. Traditionally, these equations have been solved using well-established numerical methods, such as the finite difference method (FDM) [5], finite element method [6], and spectral methods [7]. Although highly effective in many settings, these classical approaches suffer from several limitations, especially when they are applied to problems involving complex geometries, high-dimensional domains, sparse or noisy data, or discontinuous initial conditions [8].

Recently, the convergence of machine learning and computational physics has led to the emergence of novel methods intended to overcome these challenges. One such method is the physics-informed neural network (PINN), which is a deep learning framework integrating physical laws [9], expressed in the form of differential equations, directly into the training process of neural networks. Rather than learning purely from data, PINNs are trained to satisfy both observed data and the underlying governing equations of a system by incorporating the residuals of differential equations and any known initial and boundary conditions into the loss function of a neural network [3,9–11].

The core concept of PINNs is to guide the learning process using prior knowledge of a physical system [3,9–11]. By embedding these constraints into the optimization objective, a network learns to produce solutions that are not only data-consistent but also physically meaningful. This approach allows PINNs to be mesh-free, meaning they do not rely on a predefined discretization of the target domain and can work effectively even in the absence of extensive labeled data. Such properties make PINNs particularly attractive for inverse problems, real-time simulations, parameter estimation, and applications in which data are limited or expensive to obtain [9–11].

Furthermore, PINNs are highly flexible and can be applied to a wide variety of differential equations, including linear or nonlinear, deterministic or stochastic, time-dependent, and steady-state equations [10]. Additionally, PINNs can be extended to handle systems with coupled partial differential equations (PDEs), moving boundaries, or multiscale behaviors, and they can be naturally integrated with automatic differentiation tools, enabling the efficient and accurate computation of the derivatives required during training [11].

Despite their significant potential, PINNs face challenges such as sensitivity to network architecture, hyperparameter tuning,

Published under an exclusive license by KFUPM

and computational cost for complex or stiff systems [10]. Regardless, ongoing research continues to address these issues and enhance the robustness and scalability of PINNs.

PINNs have many forms and structures that vary in terms of complexity and architecture, including extended PINNs (XPINNs) [12], domain-decomposed PINNs (DD-PINNs) [13], self-adaptive PINNs (SA-PINNs) [14], and operator-learning PINNs [15]. All of these architectures retain the core principles of standard PINNs (std-PINNs), with slight modifications to their operational structure. For example, SA-PINNs are PINNs with adaptive weight balancing or residual-based sampling, whereas XPINNs are domain-decomposition-based architectures that utilize multiple neural networks for different subdomains. Similarly, DD-PINNs have a comparable structure but focus on overlapping domains and iterative coupling. Operator-learning PINNs are slightly different, as they learn mappings between function spaces, rather than just pointwise mappings, and use specialized neural operators instead of traditional dense layers (e.g., Fourier layers or attention mechanisms) [12–15].

Recent studies have extensively explored the application of PINNs to solve differential equations. Raissi et al. [9] introduced a foundational PINN framework for forward and inverse problems involving nonlinear PDEs. Subsequent studies have expanded this framework to various domains, including fluid mechanics [3], phase field modeling [10], and nonlinear dynamic systems. For example, Mattey and Ghosh [11] proposed a sequential training strategy to improve PINN performance on the Allen–Cahn and Cahn–Hilliard equations. Other developments include domain decomposition methods, such as XPINNs [12] and DD-PINNs [13], and adaptive weighting strategies, such as SA-PINNs [14].

Despite these advancements, several limitations persist. Std-PINNs often struggle with stiff and nonlinear PDEs based on issues such as gradient imbalance, sensitivity to hyperparameters, and difficulty in capturing sharp solution features [16]. In particular, equations such as the Allen–Cahn equation present significant challenges owing to their strong nonlinearity and interfacial dynamics.

The objective of this study is to evaluate the performance of std-PINNs systematically on both linear and nonlinear differential equations and to investigate the effectiveness of the backward-compatible PINN (bc-PINN) approach for overcoming these limitations. By comparing results across different problem types, we aim to highlight the strengths and weaknesses of each approach and provide insights into their real-world applicability.

We explore the theoretical foundations and practical implementations of std-PINNs for solving the damped harmonic oscillator and 1D heat equations in Sections 4 and 5, respectively. We then introduce bc-PINNs for solving the more complex 1D Allen–Cahn equation in Section 6 and 2D Allen–Cahn equation in Section 7.

The main novelty of this study lies in the provision of a unified and systematic evaluation of PINNs within a single controlled framework. Rather than demonstrating PINN performance on isolated benchmark problems, we perform consistent comparisons between std-PINNs and bc-PINNs across both ordinary differential equations (ODEs) and PDEs, including linear and nonlinear cases. By applying both methods to the same set of problems with carefully aligned architectures and training settings, our analysis isolates the effect of the learning strategy, providing clear conclusions regarding when and why bc-PINNs outperform standard formulations.

This work further contributes to the focused investigation of stiff and nonlinear phase field models, where std-PINNs struggle. By examining one- and two-dimensional Allen–Cahn equations, we demonstrate how temporal domain decomposition, solution continuity enforcement, and sequential training can significantly improve stability and accuracy. The presented results provide concrete evidence that backward-compatible training strategies are not merely incremental modifications but are essential for reliably solving nonlinear, time-dependent PDEs with sharp interface dynamics.

Finally, this report offers a practical assessment of the trade-offs associated with these methods. Although bc-PINNs achieve substantially higher accuracy on complex problems, this improvement comes at the cost of increased training time. By explicitly quantifying accuracy, convergence behavior, and computational expense relative to traditional numerical solvers, we clarify the regimes in which PINN-based approaches are advantageous or classical methods remain more efficient. This perspective helps position PINNs not as universal replacements but as complementary tools within the broader landscape of computational physics.

2. PHYSICS-INFORMED NEURAL NETWORK

Unlike traditional neural networks, which learn by comparing their outputs with correct values (ground truth), PINNs learn by incorporating physical laws described by differential equations into their loss functions, guiding the learning process toward effective solutions and eliminating the need for pre-collected training data [3]. The main feature distinguishing PINNs from traditional neural networks is their loss function, which utilizes the residuals of the differential equations governing the target problem [17]. For a general problem, the loss function is expressed as

$$MSE = MSE_{eq} + MSE_{bc} + MSE_{ic}, \tag{1}$$

where MSE, MSE_{eq}, MSE_{bc}, and MSE_{ic} represent the total mean squared error, PDE residual mean squared error, boundary condition mean squared error, and initial condition mean squared error, respectively.

The mean squared error in the initial condition is defined as

$$MSE_{ic} = \frac{1}{N_{ic}} \sum_{k=1}^{N_{ic}} \left(\hat{h}\left(x_k^{ic}, 0\right) - h_k^{ic} \right)^2, x_k^{ic} \in \Omega, \tag{2}$$

where $\hat{h}$ is the neural output prediction and h_k^{ic} is the given initial condition at $\left(x_k^{ic}, 0\right)$. The superscript $()^{ic}$ represents the initial condition. Here, Ω is an open set of $\mathbb{R}^D$ (D = 1,2,3).

- Mean squared error of the boundary condition:

$$MSE_{bc} = \frac{1}{N_{bc}} \sum_{k=1}^{N_{bc}} \sum_{d=1}^{n_d} \left(\hat{h}^{(d-1)} \left(x_k^{bc}, t_k^{bc} \right) - h^{(d-1)} \right)^2 \left(x_k^{bc}, t_k^{bc} \right) \in \Gamma \times (0, T], \tag{3}$$

where $\hat{h}^{(d-1)} \left(x_k^{bc}, t_k^{bc} \right)$ are the neural output predictions, and $h^{(d-1)}$ is the given boundary condition at $\left(x_k^{bc}, t_k^{bc} \right)$. The superscript $()^{bc}$ represents the boundary condition. Here, Ω is an open set of $\mathbb{R}^D$ (D = 1,2,3), and Γ is the boundary of Ω.

- Mean squared error due to the residual of the PDE:

$$R := \hat{h}_t - F \left(\hat{h}, \hat{h}_x^{(1)}, \hat{h}_x^{(2)}, \ldots, \hat{h}_x^{(m)} \right)$$

$$\text{MSE}_{eq} = \frac{1}{N_{eq}} \sum_{k=1}^{N_{eq}} \left(R \left(x_k^{eq}, t_k^{eq} \right) \right)^2, \left(x_k^{eq}, t_k^{eq} \right) \in \Omega \times (0, T], \tag{4}$$

where R is the residual of the PDE at the collocation point $\left(x_k^{eq}, t_k^{eq} \right)$. The superscript $()^{eq}$ represents the residual of the PDE. Here, Ω is an open set of $\mathbb{R}^D$ (D = 1,2,3).

These loss equations are general, and for real-world applications, there may be some additional loss terms. For example, the boundary condition loss could be further extended to Dirichlet or Neumann boundary condition loss [18], and the initial condition loss may be further extended to interface jump conditions for the primary variable or flux loss [18]. Furthermore, one could add a loss term to incorporate the given data, if applicable. This hybrid approach of using both data-driven and physical laws may further increase the accuracy of PINNs [11].

The loss function of a PINN is minimized using appropriate optimization algorithms. Many optimization algorithms have been used to minimize loss, including stochastic gradient descent, Adam, and limited Broyden–Fletcher–Goldfarb–Shanno (L-BFGS), with the Adam optimizer being the most popular owing to its fast convergence [8]. The overall loss that must be minimized during training can be customized according to the target problem. Depending on the available data and characteristics of the problem, the total loss may encompass either data-driven or physics-informed components. In the case of inverse problems, where the fundamental physics is partially understood but enhanced by experimental data, it is advantageous to incorporate both data-driven and physics-informed loss components [10]. By minimizing this combined loss function, the neural network can concurrently determine the optimized physical parameters $\lambda*$ and network parameters $\theta*$ to approximate the solution. This process can be expressed mathematically as follows:

$$\theta*, \lambda* = argmin \mathcal{L}(\theta; \lambda), \; where \; \mathcal{L} = \mathcal{L}_D + w\mathcal{L}_P, \tag{5}$$

where $\theta*$, $\lambda*$ represent the optimal network and physical parameters, respectively. $\mathcal{L}_D$ and $\mathcal{L}_P$ are the data-driven and physically informed loss terms, respectively, and w is a parameter used to balance the contributions of each term [3,9,10].

The selection of network architectures in this study was performed to balance model expressiveness and computational efficiency. For simpler problems, such as the damped harmonic oscillator and 1D heat equation, a relatively shallow network (three hidden layers with 32 neurons) is sufficient to capture solution behavior. However, more complex problems, such as the 2D Allen–Cahn equation, require deeper and wider networks (six hidden layers with 64 neurons) to represent higher-dimensional and nonlinear solution spaces. The batch size was selected implicitly based on the collocation points sampled across the domain, and training was performed using full-batch optimization. The Adam optimizer was used for initial convergence, while L-BFGS with box constraints (L-BFGS-B) was employed for fine-tuning. Training was terminated after a fixed number of iterations to ensure consistency and reproducibility across experiments.

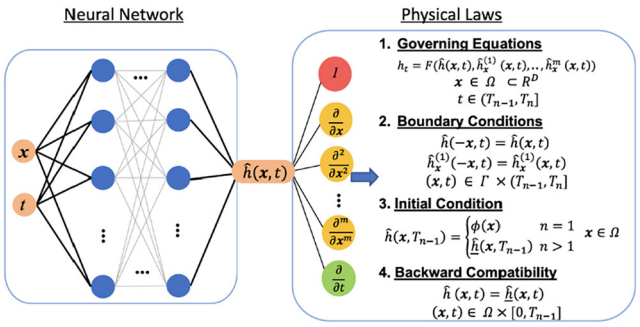

Figure 1. Schematic of the bc-PINN framework for a time segment $[T_{n-1}, T_n]$. The neural network is retrained in each time segment while enforcing solution continuity by using the solution from the previous segment. The initial condition is $h(x, 0) = g(x)$ for the first segment and $h(x, T_{n-1}) = h(x, T_{n-1})$ for all subsequent segments

3. BACKWARD-COMPATIBLE SEQUENTIAL PINN METHOD

Bc-PINNs are essentially std-PINNs with the added ability to solve PDEs over time by re-training a single neural network across successive time segments [11,19]. Std-PINNs often struggle to handle difficult PDEs, such as the Allen–Cahn and Cahn–Hilliard equations, owing to their strong nonlinearities, higher-order derivatives, and sharp interface dynamics. Bc-PINNs overcome this issue by breaking the problem into manageable temporal chunks and ensuring solution continuity across segments [11].

In bc-PINNs, the time domain $[0, T]$ is discretized into n_{max} segments as follows:

$$[T_0 = 0, T_1 \,],] \, T_1, T_2 \ldots, [\, T_{n-1}, T_n \,], ,] \, T_{n_{max}-1}, T_{n_{max}} = T], \tag{6}$$

where the nth segment is denoted as

$$\Delta T_n = [T_{n-1}, T_n], n = 1, \ldots, n_{max}.$$

To ensure the continuity of the solution, all previous solutions are enforced, resulting in a new loss function given by

$$\begin{aligned} &\text{MSE}_{\Delta T_n} = w_i \, \text{MSE}_{ic} \left(\mathbf{x}_k^{ic}, T_{n-1} \right) + w_b \, \text{MSE}_{bc} \left(\mathbf{x}_k^{bc}, t_k^{bc} \right) \\ &+ w_r \, \text{MSE}_{eq} \left(\mathbf{x}_k^{r}, t_k^{r} \right) + w_s \, \text{MSE}_S \left(\mathbf{x}_k^{s}, t_k^{s} \right), \\ &n = 2, \ldots, n_{\max} \mathbf{x}_k^{ic} \in \Omega, \left(\mathbf{x}_k^{bc}, t_k^{bc} \right) \in \Gamma \times (T_{n-1}, T_n] \\ &\left(\mathbf{x}_k^{r}, t_k^{r} \right) \in \Omega \times (T_{n-1}, T_n], \left(\mathbf{x}_k^{s}, t_k^{s} \right) \in \Omega \times [0, T_{n-1}). \end{aligned} \tag{7}$$

In this context, $\left(\mathbf{x}_k^i, T_{n-1}\right)$ denotes the collection of points at which the error related to the initial condition is evaluated, whereas $\left(\mathbf{x}_k^{bc}, t_k^{bc}\right)$ signifies the collection of points at which the error concerning the boundary conditions is assessed within the time interval$(T_{n-1}, T_n]$. Additionally, the derivation is reduced from the previously acquired solution recorded at grid points $\left(\mathbf{x}_k^s, t_k^s\right)$ to ensure solution continuity. To learn more about the derivation of the loss function, we refer readers to [11].

Two additional techniques were adopted to ensure robust accuracy: initial condition-guided learning (ICGL) and transfer-learning-based acceleration (TL) [11].

- ICGL

ICGL separates training into two stages for each time segment. In the first stage, the neural network is trained solely to align with the initial condition of a specific time segment, utilizing a small fraction of the total iterations [11]. Consequently, the loss function for ICGL (MSE_{SI}) is defined as follows:

$$\mathrm{MSE}_{SI} = \frac{1}{N_{SI}} \sum_{k=1}^{N_{SI}} \left(\bar{h}\right), \left(\mathbf{x}_k^{SI}, t_k^{SI}\right) \in \Omega \times \left(T_{n-1}, T_n\right], \tag{8}$$

where $\hat{h}$ are the neural network predictions, and $\bar{\bar{h}}_k^{SI}\left(\mathbf{x}_k^{SI}, T_{n-1}\right)$ is the known solution from the neural network in the previous time step T_{n-1}.

- TL

In this stage, the obtained weights (network parameters $\theta*$) are transformed into initial weights for the neural network. This approach is effective because the solution of a PDE within a brief time interval is anticipated to resemble the initial condition closely if the interval is limited [11].

4. DAMPED HARMONIC OSCILLATION

The damped harmonic oscillation equation describes the motion of a system in which an object experiences a restoring force proportional to its displacement and a resistive force proportional to its velocity. This type of motion commonly occurs in mechanical and electrical systems [20], where energy gradually dissipates over time as a result of friction, air resistance, or other damping effects.

4.1. Equation Selection. Our first priority in equation selection is simplicity, as the capabilities of PINNs should be evaluated utilizing relatively simple equations. The damped harmonic oscillation equation is an ODE that is mathematically expressed as

$$m\frac{d^2u}{dt^2} + \mu\frac{du}{dt} + ku = 0, \tag{9}$$

where m is the mass, $c$is the damping coefficient, $k$is the spring constant, and $x\,(t)$ is the displacement over time. Depending on the damping strength, the system may exhibit underdamped, critically damped, or overdamped behaviors, each of which yields distinct dynamic characteristics [21].

4.2. PINN Training for ODEs. For the damped harmonic oscillation problem, a relatively shallow network architecture consisting of three hidden layers with 32 neurons per layer was selected. This architecture was selected because the governing equation is a low-dimensional ODE with smooth temporal behavior and limited nonlinear complexity. Preliminary experiments with deeper architectures yielded negligible improvements in prediction accuracy, while substantially increasing computational cost and training time.

Therefore, the selected architecture represents a favorable balance between computational efficiency and approximation capability. Unlike highly nonlinear PDEs, simple ODEs generally do not require large representational capacity to approximate smooth solutions accurately. Similar observations regarding reduced architectural requirements for low-dimensional problems have been reported in the PINN literature [3,9,11].

The hyperbolic tangent (tanh) activation function was used in all hidden layers, with no activation function being applied to the output layer, which is a common approach to solving differential equations using PINNs [3]. Other activation functions, such as a rectified linear unit, could also be used; however, tanh was selected for its smooth curve within the range of -1 to 1.

The std-PINN model was trained for 5,000 iterations using the Adam optimizer. Adam was selected based on its ability to handle imbalanced gradients, adapt to varying curvatures in the loss landscape, and provide stable convergence for physics-informed optimization problems [18]. We relied solely on the physics-informed loss, without using training data. In our model, we adopted the following initial conditions:

$$u\,(0) = 1, \quad \frac{du}{dt}\,(0) = 0. \tag{10}$$

For Eq. (10), we set $m = 1$, $\mu = 4$, and $k = 400$. The model was trained over the domain $t \in [0, 1]$. We did not define boundary conditions for this problem, considering the nature of the selected equation.

4.3. Results . An exact solution is required to evaluate the accuracy of the PINN results. Fortunately, the exact solution for the selected problem is known and is given by

$$u\,(t) = e^{-\delta t}\,(2Acos\,(\emptyset + \omega t)), \tag{11}$$

where

$$\omega = \sqrt{\omega_o^2 - \delta^2}, \tag{12}$$

$$\emptyset = \tan^{-1}\left(-\frac{\delta}{\omega}\right), \tag{13}$$

$$A = \frac{1}{2\cos\,(\emptyset)}. \tag{14}$$

In Eq. (12), we set $\omega_o = 20$, and in Eqs. (11) to (13), we set $\delta = 2$. The resulting solution is illustrated in Fig. 2.

The PINN demonstrates excellent agreement with the exact solution while relying solely on the governing physical laws.

5. HEAT EQUATION

The heat equation is a fundamental PDE that describes the heat distribution (or temperature) in a given domain by modelling thermal diffusion as heat flows from regions of higher temperature to regions of lower temperature.

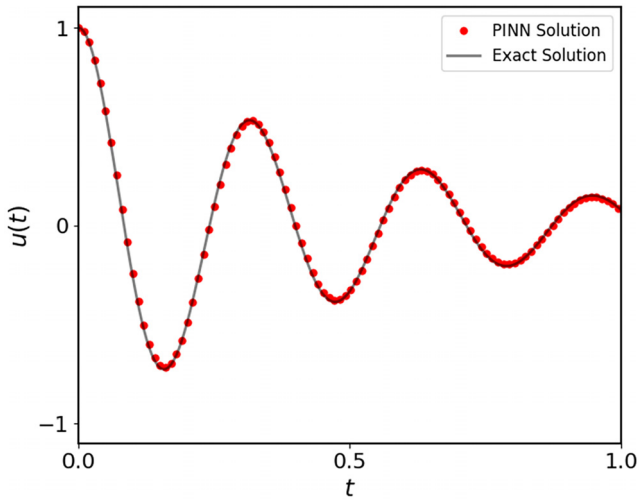

Figure 2. Exact solution and std-PINN prediction for the damped harmonic oscillation equation

5.1. Equation Selection. After examining the PINN's ability to solve ODE problems, we now focus on PDE problems. First, a linear heat equation (PDE) is used to evaluate the capability of the model to solve relatively simple PDEs. Later, this analysis is extended to a more complex nonlinear PDE: the Allen–Cahn equation. In one spatial dimension, the heat equation is given by

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}, \tag{15}$$

where $u(x,t)$ represents the temperature at position $x$and time t, and α is the thermal diffusivity constant of the material. The heat equation is widely used in physics and engineering to simulate heat conduction in solids and serves as a prototype for other diffusion-type problems in fields such as biology, finance, and environmental science.

5.2. PINN Training for the Heat Equation. The same network architecture used for the damped harmonic oscillator was retained for the 1D heat equation to evaluate whether a relatively lightweight PINN can accurately solve a linear PDE problem. Because the heat equation exhibits smooth, diffusion-dominated behavior, without strong nonlinearities or sharp interfaces, a shallow architecture with three hidden layers and 32 neurons per layer was found to be sufficient for stable convergence and accurate approximation.

Increasing the number of neurons or hidden layers during preliminary testing yielded only marginal accuracy improvements, while increasing the computational cost substantially. Therefore, a smaller architecture was maintained for efficiency and consistency across simpler benchmark problems.

The tanh activation function was used in all hidden layers, with no activation function applied to the output layer. The Adam optimizer was selected for the same reasons discussed in the previous section.

For this problem, we defined the following conditions:

- Initial conditions:

$$u(x,0) = sin\left(\frac{\pi x}{L}\right), 0 \le x \le L. \tag{16}$$

- Boundary conditions:

$$u(0,t) = 0, u(L,t) = 0, t \ge 0. \tag{17}$$

Here, $u(x,0)$ is the temperature at position x at the initial time $t = 0$, and$L = 1$ in Eqs. (16) and (17). We consider the time interval over the domain $t \in [0,1]$. The boundary conditions were set to zero because $sin(0) = 0$ and $sin(\pi) = 0$.

5.3. Results . As mentioned in the previous section, to evaluate the accuracy of results, an exact solution is required, which is given by

$$u(x,t) = e^{-\alpha\left(\frac{\pi}{L}\right)^2 t} sin\left(\frac{\pi x}{L}\right), \tag{18}$$

where, $L = 1$ and $t \in [0,1]$. The results for the heat equation are illustrated in Fig. 3.

The model accurately predicts the solution to a more complex problem.

6. 1D ALLEN–CAHN EQUATION

The 1D Allen–Cahn equation is a nonlinear PDE used to describe the evolution of phase separation in materials science, particularly in the context of two-phase systems, such as solid–solid or liquid–liquid mixtures.

6.1. Equations . After observing the std-PINN's capability to learn the linear 1D heat PDE, we increased the level of challenge and tested the PINN's ability to learn the nonlinear 1D Allen–Cahn PDE. This PDE is given by

$$\frac{\partial u}{\partial t} = \epsilon^2 \frac{\partial^2 u}{\partial x^2} - f(u), \tag{19}$$

where $u(x,t)$ is an order parameter that typically varies between two stable phases, $\epsilon > 0$ is a small parameter related to the interface thickness between phases, and$f(u) = F''$ is the derivative of a double-well potential.

6.2. Std-PINN Application. Before evaluating the bc-PINN, we first applied the std-PINN model to this nonlinear PDE. The network architecture, activation function, number of neurons, and optimizer settings were the same as those described in the previous two sections. The following conditions were applied.

- Initial conditions:

$$u(x,0) = x^2 cos(\pi x), -1 \le x \le 1, \tag{20}$$
$$f(u) = 5\left(u - u^3\right). \tag{21}$$

- Boundary conditions:

$$u(-1,t) = u(1,t), \tag{22}$$
$$u_x(-1,t) = u_x(1,t). \tag{23}$$

By substituting Eq. (21) into Eq. (19), we obtain the following final equation:

$$u_t = \epsilon^2 u_{xx} + 5u - 5u^3, \quad t \in [0,1]. \tag{24}$$

For this problem, we set $\epsilon = 0.01$.

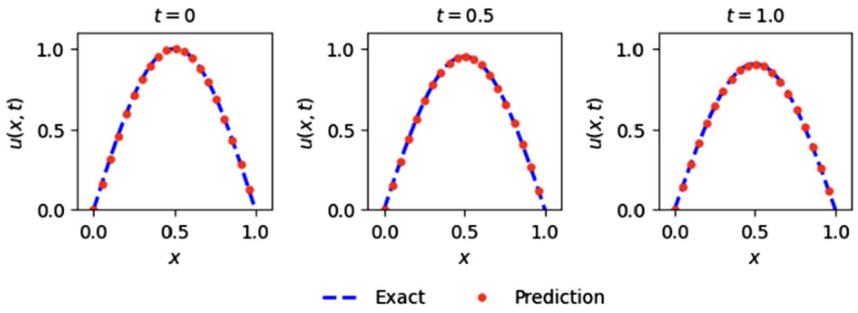

Figure 3. Exact solution and std-PINN prediction for the heat equation at t = 0, t = 0.5, and t = 1

The exact solution was obtained from the FDM and compared with the std-PINN prediction after training for 5,000 iterations. The results are presented in Fig. 4.

The std-PINN model failed to learn the Allen–Cahn equation because of its strong nonlinearity and sharp interfacial transitions [11,16], as shown in Fig. 5.

6.3. Introducing the bc-PINN. In this section, we apply the bc-PINN model to the 1D Allen–Cahn PDE and observe whether it can learn the PDE and provide an accurate solution. Compared with the previous ODE and linear PDE problems, the Allen–Cahn equation represents significantly greater learning difficulty owing to its strong nonlinearities, sharp interfacial transitions, and increased stiffness. Consequently, a larger network capacity and a hybrid optimization strategy are required to stabilize training and accurately capture the solution dynamics. While [10] employed large architectures with approximately 100 neurons for phase field models, we selected architectures to balance solution accuracy, convergence stability, and computational efficiency for each specific problem, rather than adopting a single fixed configuration across all cases.

For the implemented bc-PINN architecture, two neurons were used in the input layer because the inputs consisted of both spatial coordinates and time, 20 neurons were used in each of the four hidden layers, and one neuron was used in the output layer. All hidden layers employed the tanh activation function for the reasons explained in the previous sections. For optimization, a hybrid approach was implemented, utilizing both the Adam and L-BFGS-B algorithms. Adam is an adaptive learning rate optimization method that is generally effective at rapidly reducing initial loss [8], whereas L-BFGS-B is a limited-memory quasi-Newton method that tends to perform well on problems with smooth objective functions [22] and can converge to a higher precision than methods such as Adam. In this implementation, the Adam optimizer was used for rapid initial convergence, while L-BFGS-B was applied periodically for fine-tuning, potentially resulting in a lower final loss. The model was trained for 5,000 ADAM iterations, with 10,000 LBFGS iterations every 1,000 ADAM iterations, across 10 time steps.

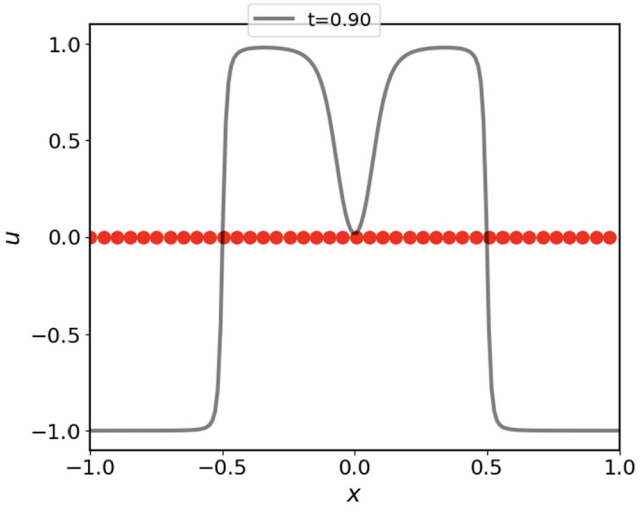

Figure 4. Red circles represent the std-PINN predictions, while the grey line represents the exact solution using the FDM with t = 0.9

Solving the same Allen–Cahn PDE introduced in the previous subsection using the trained bc-PINN model yielded the solution presented in Fig. 5.

The solution for $t = 0$, which represents the initial condition, demonstrates that the bc-PINN correctly matches the prescribed initial profile. The solution for $t = 0.5$ represents the intermediate evolution of the phase field, where the bc-PINN accurately captures the nonlinear interface dynamics. The solution for $t = 1$ illustrates the long-term behavior of the solution and confirms the bc-PINN's ability to maintain stability and accuracy over extended time horizons. The main factors behind the bc-PINN's ability to predict the solution much more accurately than the std-PINN are the techniques used to break down the time interval into small, tolerable segments and ensure the solution's continuity [11], as described in Section 3. Because the one-dimensional Allen–Cahn equation is more complex than the previous problems, a notable increase in the training time is observed, with approximately 4 hours of training.

7. 2D ALLEN–CAHN EQUATION

As described in Section 6, the Allen–Cahn equation is a fundamental PDE used to model phase separation processes and the evolution of interfaces in multiphase systems. This PDE was originally introduced in one spatial dimension to describe the motion of antiphase boundaries in binary alloys and has since been generalized to higher dimensions for a broader range of applications in materials science, image processing, and geometric flows. Furthermore, numerical approximation and analysis become significantly more challenging in two dimensions due to the presence of anisotropic interface motion, emergence of singularities, and increased computational cost of resolving fine interfacial structures, especially in the sharp interface limit $\epsilon \to 0$. Therefore,

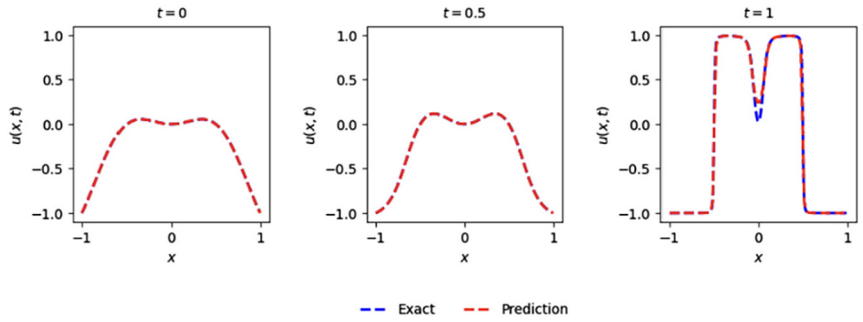

Figure 5. Exact solution and bc-PINN prediction for the 1D Allen–Cahn equation for t = 0, t = 0.5, and t = 1

while the 1D Allen–Cahn equation provides a useful theoretical and computational testbed, the 2D version captures a far broader and more realistic range of physical behaviors, making it a central object of study in modern phase field modeling.

7.1. Equation Selection. In two dimensions, the Allen–Cahn equation is given by

$$\frac{\partial u}{\partial t} = \epsilon^2 \nabla^2 u - f(u). \tag{25}$$

Here, the Laplacian $\nabla^2 u$ governs the diffusion in two spatial dimensions, and the geometry of the evolving interface plays a critical role. Unlike in the 1D case, where the phase boundaries are isolated points, in the 2D case, these boundaries become curves whose motion is governed by the mean curvature flow. This higher-dimensional setting supports the development of intricate patterns, interface instabilities, and topological changes, such as the merging or annihilation of domains and phenomena that are absent in the 1D scenario.

7.2. PINN Training for 2D Allen–Cahn. For the 2D Allen–Cahn equation, a deeper and wider architecture consisting of six hidden layers with 64 neurons per layer was employed. Compared with the previous benchmark problems, the 2D Allen–Cahn equation contains significantly more complex spatial dynamics, nonlinear interface evolution, and higher-dimensional feature interactions. Consequently, the network requires increased representational capacity to approximate the solution manifold accurately.

Preliminary experiments with smaller architectures resulted in slow convergence and reduced prediction quality near sharp interfaces. Increasing the depth and width of the network improved the ability of the model to capture fine spatial structures and maintain stability throughout training. Therefore, a larger architecture was selected as a compromise between computational feasibility and approximation accuracy.

The tanh activation function was used in all hidden layers, with no activation function applied to the output layer. For the optimization process, the hybrid approach introduced in the previous section was employed.

For this problem, the following conditions were applied[4]:

- Initial conditions:

$$u(x, y, 0) = sin(4\pi x)\, cos(4\pi y), 0 \le x, y \le 1, \tag{26}$$

$$f(u) = u - u^3. \tag{27}$$

- Boundary conditions:

$$u^{(d-1)}(x, y, t) = u^{(d-1)}(-x, -y, t), d = 1, 2. \tag{28}$$

By substituting Eq. (27) into Eq. (25), we obtain the following final equation:

$$u_t = \epsilon^2 (u_{xx} + u_{yy}) + u - u^3, t \in [0, 1]. \tag{29}$$

For this problem, we set $\epsilon = 0.01$.

7.3. Results . After training the model for 10,000 ADAM iterations with 5,000 LBFGS iterations at every 1,000 ADAM iterations with 20 time steps, the results were as presented in Fig. 6.

The bc-PINN successfully captured the solution of the 2D Allen–Cahn equation, accurately resolving the evolution of complex interfacial structures over time. The predicted phase field profiles remain consistent with the reference solution across all reported time horizons, indicating that the sequential training strategy effectively preserved the solution continuity and stability in higher dimensions. However, this improved accuracy came at a high computational cost. In this case, the bc-PINN required approximately 50 h of training, whereas the corresponding finite-difference-based numerical solver completed the same simulation in less than one hour on comparable hardware. This substantial increase in the computational time arises from repeated network retraining over multiple time segments, high-dimensional collocation sampling, and the use of hybrid optimization schemes involving both Adam and L-BFGS. These results highlight a key tradeoff: while bc-PINNs provide a flexible, mesh-free framework capable of handling nonlinear and high-dimensional PDEs, their practical deployment for large-scale problems remains limited by poor training efficiency compared with classical numerical methods.

8. ARCHITECTURE SENSITIVITY AND ROBUSTNESS

To evaluate the robustness of the selected network architectures further, preliminary sensitivity experiments were conducted using different numbers of hidden layers and neurons for each class of differential equations. However, for the damped harmonic oscillator and 1D heat equation, increasing the architecture size beyond three hidden layers with 32 neurons yielded only marginal improvements in prediction accuracy, while substantially increasing computational cost and training time.

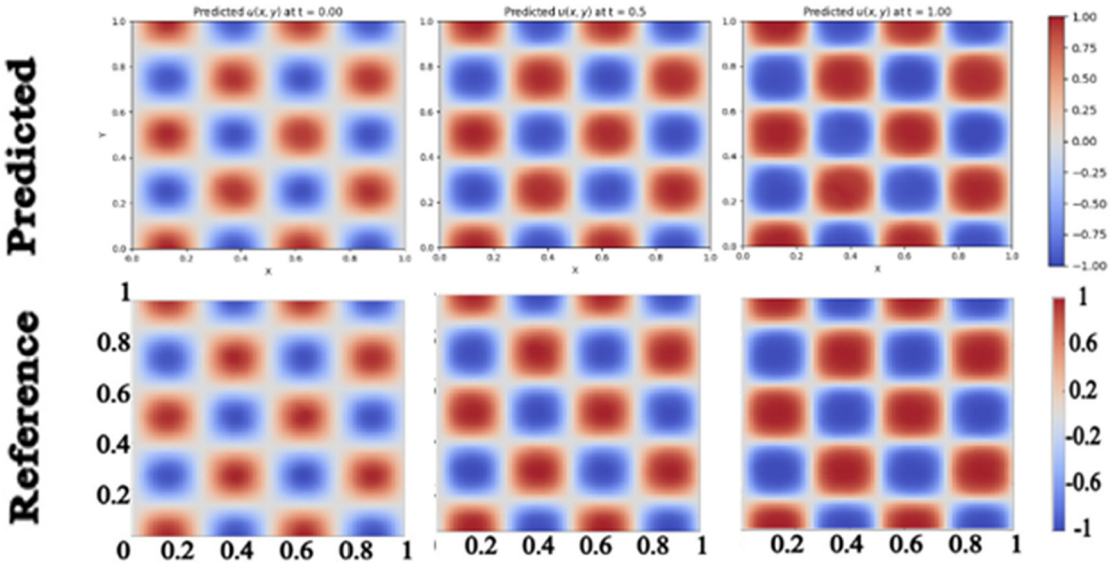

Figure 6. Comparison between the reference solution and bc-PINN prediction for the 2D Allen–Cahn equation at t = 0, 0.5, and 1

By contrast, the nonlinear Allen–Cahn equations exhibit significantly greater sensitivity to network capacity. Smaller architectures frequently struggle to capture sharp interfacial behavior and nonlinear solution dynamics accurately, particularly in 2D cases. Increasing both the depth and width of the network improved convergence stability, reduced oscillatory prediction behavior, and enhanced the ability of the model to represent fine spatial structures.

These observations are consistent with those of previous PINN studies reporting that nonlinear and high-dimensional PDEs generally require greater representational capacity than smooth low-dimensional ODEs[3–6]. Consequently, the architectures adopted in this study were selected according to the complexity of each differential equation, rather than using a single fixed architecture across all cases.

9. DISCUSSION

The results obtained from the four representative benchmark problems provide a comprehensive assessment of the capabilities and limitations of PINNs for solving differential equations of varying complexity. For the damped harmonic oscillator and 1D heat equation, std-PINNs demonstrate high accuracy and stability, successfully approximating exact analytical solutions. However, for nonlinear and more complex PDEs, particularly the 1D and 2D Allen–Cahn equations, std-PINNs fail to capture correct solution behavior. These problems require substantially larger network architectures and more sophisticated optimization strategies to achieve stable convergence, highlighting the strong dependence of PINN performance on model capacity and problem complexity. This limitation is primarily attributable to the strong nonlinearity, sharp interface dynamics, and increased sensitivity of training parameters.

Table 1. *PINN architectures used for each differential equation.*

Problem	Hidden Layers	Neurons Layer	Optimizer
Damped Oscillator	3	32	Adam
1D Heat	3	32	Adam
1D Allen–Cahn	4	20	Adam + L-BFGS-B
2D Allen–Cahn	6	64	Adam + L-BFGS-B

By contrast, bc-PINNs significantly improve performance by decomposing the time domain into smaller segments and enforcing continuity across segments. This approach enables the model to handle temporal dependencies and nonlinearities more effectively, resulting in accurate predictions for both the 1D and 2D Allen–Cahn equations. When compared with the results reported in previous studies[4], our findings are generally consistent, confirming that sequential training strategies enhance the robustness and accuracy of PINNs for complex PDEs. However, this improvement comes at the cost of increased computation time, particularly for high-dimensional problems.

10. CONCLUSION

This study demonstrated the effectiveness and versatility of PINNs for solving various types of differential equations, including ODEs, linear PDEs, and nonlinear PDEs. The std-PINN was successfully applied to simpler equations, such as the damped harmonic oscillator and 1D heat equations, achieving high accuracy when approximating known solutions without relying on training data. However, when applied to the more complex and nonlinear 1D Allen–Cahn equation, the std-PINN struggled to converge to an accurate solution based on the nonlinearity and sharp interface dynamics of the equation.

To overcome the limitations observed with std-PINNs, specifically their poor convergence for nonlinear and stiff PDEs, sensitivity to sharp interface dynamics, and degradation of accuracy over long time horizons, a bc-PINN was introduced. Std-PINNs struggle to resolve the Allen–Cahn equation owing to its strong nonlinearity and rapidly evolving interfaces, often becoming trapped in suboptimal solutions. By segmenting the

temporal domain and explicitly enforcing solution continuity between consecutive time intervals, bc-PINNs can alleviate these issues significantly and enable stable long-term integration. Additionally, the use of ICGL improves convergence at the start of each time segment, while transfer learning accelerates training by reusing previously learned network weights. Together, these strategies allow the bc-PINN to maintain its accuracy and stability over extended time intervals, resulting in a marked improvement over the std-PINN for solving the Allen–Cahn equation.

Overall, this study illustrates the significant potential of PINNs as mesh-free, data-efficient solvers for differential equations, particularly when architectural improvements such as bc-PINNs are considered. Our results further demonstrate that selecting an appropriate network architecture based on the dimensionality and nonlinearity of the governing equation is critical for achieving stable and accurate PINN performance. These results reinforce the value of PINNs in computational science and suggest promising directions for future research, including the application of PINNs to high-dimensional systems, coupled PDEs, and real-world inverse problems.

AFFILIATIONS AND AUTHOR DETAILS

Undergraduate Author

Osama Hawladar – *Department of Aerospace Engineering, King Fahd University of Petroleum and Minerals, Dhahran, 31261 Saudi Arabia*; 0009-0008-1209-1575
Email: s202252440@kfupm.edu.sa

Corresponding Author

Abdullah Shah – *Department of Mathematics, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia; Interdisciplinary Research Centre for Refining & Advanced Chemicals, KFUPM, Dhahran 31261, Saudi Arabia*; 0000-0002-0337-1216
Email: abdullah.shah.1@kfupm.edu.sa

ACKNOWLEDGEMENTS

This work was conducted during the authors' visit to the Interdisciplinary Center for Scientific Computing (IWR), Heidelberg University, in the summer of 2025, hosted by Professor Peter Bastian. The authors gratefully acknowledge the support of King Fahd University of Petroleum and Minerals (KFUPM). O. Hawladar acknowledges support from the KFUPM Global Exchange Fellowship Program, and A. Shah acknowledges support from the KFUPM Dean of Research through the Ibn Battuta Outbound Program.

CODE AVAILABILITY

All codes and data are available at: https://github.com/Physics-informed-neural-networks

REFERENCES

(1) Einstein, A. & Infeld, L. On the motion of particles in general relativity theory. *Can. J. Math.* **1**, 209–241 (1949).

(2) Jain, S. C. Simple solutions of the partial differential equation for diffusion (or heat conduction). *Proc. R. Soc. Lond. A* **243**, 359–374 (1958).

(3) Cai, S., Mao, Z., Wang, Z. et al. Physics-informed neural networks (PINNs) for fluid mechanics: a review. *Acta Mech. Sin.* **37**, 1727–1738 (2021).

(4) Belova, M. et al. Study of the impact of loads on the deformation of building structures using differential equations. *WSEAS Trans. Syst.* **23**, 398–408 (2024).

(5) Tourin, A. Introduction to finite difference methods. *Optimal Stochastic Control, Stochastic Target Problems, and Backward SDE*, **29**, 201–212 (2013).

(6) Jagota, V., Sethi, A. P. S. & Kumar, K. Finite element method: an overview. *Walailak J. Sci. Technol.* **10**, 1–8 (2013).

(7) Chen, L. Q. & Shen, J. Applications of semi-implicit Fourier-spectral method to phase field equations. *Comput. Phys. Commun.* **108**, 147–158 (1998).

(8) Quarteroni, A. Numerical Models for Differential Problems, 3rd edn. *Springer* (2017).

(9) 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. *J. Comput. Phys.* **378,** 686–707 (2019).

(10) Li, W., Fang, R., Jiao, J., Vassilakis, G. N. & Zhu, J. Tutorials: physics informed machine learning methods of computing 1D phase-field models. *APL Mach. Learn.* **2**, 031101 (2024).

(11) Mattey, R. & Ghosh, S. A novel sequential method to train physics informed neural networks for Allen Cahn and Cahn Hilliard equations. *Comput. Methods Appl. Mech. Eng.* **390**, 114474 (2022).

(12) Jagtap, A. D. & Karniadakis, G. E. Extended physics-informed neural networks (XPINNs): a generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Commun. Comput. Phys.* **28,** 2002–2041 (2020).

(13) Gu, L. et al. Physics-informed neural networks with domain decomposition for the incompressible Navier–Stokes equations. *Phys. Fluids* **36,** 021914 (2024).

(14) Zhang, R. et al. BO-SA-PINNs: self-adaptive physics-informed neural networks based on Bayesian optimization for automatically designing PDE solvers. *arXiv:2504.09804* (2025).

(15) Gonon, L. et al. An overview on machine learning methods for partial differential equations: from physics informed neural networks to deep operator learning. *arXiv:2408.13222* (2024).

(16) Fernández de la Mata, F., Gijón, A., Molina-Solana, M. & Gómez-Romero, J. Physics-informed neural networks for data-driven simulation: advantages, limitations, and opportunities. *Physica A* **610**, 128415 (2023).

(17) Psaros, A. F., Kawaguchi, K. & Karniadakis, G. E. Meta-learning PINN loss functions. *J. Comput. Phys.* **458**, 111121 (2022).

(18) Song, Y., Yuan, X. & Yue, H. The ADMM-PINNs algorithmic framework for nonsmooth PDE constrained optimization: a deep learning approach. *SIAM J. Sci. Comput.* **46**, C659–C687 (2024).

(19) Zhou, C. et al. Physics-informed neural networks with complementary soft and hard constraints for solving complex boundary Navier–Stokes equations. arXiv:2411.08122 (2024).

(20) Stevens, K. W. H. The wave mechanical damped harmonic oscillator. *Proc. Phys. Soc.* **72**, 1027 (1958).

(21) Gupta, R., Gupta, R. & Rajput, S. Analysis of damped harmonic oscillator by matrix method. *Int. J. Res. Anal. Rev.* **5**(4), 479–484 (2018).

(22) Bemporad, A. An L-BFGS-B approach for linear and nonlinear system identification under $\ell 1$ and group-lasso regularization. *IEEE Trans. Autom. Control* **70**, 4857–4864 (2025).