Journal of Computational Science 90 (2025) 102637

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

DP-PINN+: A Dual-Phase PINN learning with automated phase division

Da Yan, Ligang He *

Department of Computer Science, University of Warwick, Coventry, UK

ARTICLE INFO

Keywords:
Physics-Informed Neural Networks
Partial differential equations
Model training
Data sampling

ABSTRACT

Physics-Informed Neural Networks (PINNs) are a promising application of deep neural networks for the numerical solution of nonlinear partial differential equations (PDEs). However, it has been observed that standard PINNs may not be able to accurately fit all types of PDEs, leading to poor predictions for specific regions in the domain. A common solution is to partition the domain by time and train each time interval separately. However, this approach leads to the prediction errors being accumulated over time, which is especially the case when solving "stiff" PDEs. To address these issues, we propose a new PINN training scheme, called DP-PINN+ (Dual-Phase PINN+). DP-PINN+ divides the training into two phases based on a carefully chosen time point t_s. The phase-1 training aims to generate the accurate solution at t_s, which will serve as the additional intermediate condition for the phase-2 training. The method for determining the optimized value of t_s is proposed in this paper. Further,new sampling strategies are proposed to enhance the training process. These design considerations improve the prediction accuracy significantly. We have conducted the experiments to evaluate DP-PINN+ with both "stiff" and non-stiff PDEs, including 1D Burger's Equation, 1D Allen–Cahn Equation, 2D and 3D Navier–Stokes Equations (i.e., 2D cylinder wake and 3D unsteady Beltrami flow). We compared DP-PINN+ with the state-of-the-art PINNs in literature, including Time Adaptive PINN, SA-PINN, bc-PINN and NSFNets.The results show that the solutions predicted by DP-PINN+ exhibit significantly higher accuracy. This paper is extended from our conference paper published in ICCS2024 Yan and He (2024) [1].

1. Introduction

Traditional physics-based numerical methods [2–4] have had great success in solving partial differential equations (PDEs) for a variety of scientific and engineering problems. While these methods are accurate, they are computationally intensive for complex problems such as nonlinear partial differential equations and often require problem-specific techniques. Over the past decade, data-driven methods have gained significant attention in various areas of science and engineering. These methods can identify highly nonlinear mappings between inputs and outputs, potentially replacing or augmenting expensive physical simulations. However, typical data-driven deep learning methods tend to ignore a physical understanding of the problem domain [5].

In order to incorporate physical priors into the model training, a new deep learning technique, Physics-Informed Neural Networks (PINNs) [6], has been proposed. Propelled by vast advances in computational capabilities and training algorithms, including the availability of automatic differentiation methods [7], PINNs combine the idea of using neural networks as generalized function approximator for solving PDEs [8] and the idea of using the system of PDEs as physical priors to constrain the output of the neural networks, which makes neural networks a new and effective approach to solving PDEs.

The original PINN algorithm proposed in [6], hereafter referred to as the "standard PINN", is effective in estimating solutions that are reasonably smooth with simple boundary conditions (e.g., the specific boundary values are given), such as the viscous Burger's equation, Poisson's equation, Schrödinger's equation and the wave equation. On the other hand, it has been observed that the standard PINN has the convergence and accuracy problems when solving "stiff" PDEs [9] such as the nonlinear Allen–Cahn equation, where solutions contain sharp space transitions or fast time evolution.

To solve these problems, numerous methods have been proposed recently, including bc-PINN [10], the time adaptive approach [11] and SA-PINN [12]. Among them, bc-PINN is especially noteworthy for its simple and intuitive philosophy. Since the stiff PDE has sharp, fast space/time transitions [13], it is hard to predict a domain as a whole. The key idea of bc-PINN is to retrain the same neural network for solving the PDE over successive time segments while satisfying the already obtained solutions for all previous time segments.

However, our analysis reveals that the training scheme in bc-PINN may lead to a progressive accumulation of prediction errors across successive time segments. Our explanation for this phenomenon is that after bc-PINN trains the solutions in a time segment, the trained

* Corresponding author.
E-mail addresses: da.yan@warwick.ac.uk (D. Yan), ligang.he@warwick.ac.uk (L. He).

https://doi.org/10.1016/j.jocs.2025.102637
Received 15 October 2024; Received in revised form 5 May 2025; Accepted 21 May 2025
Available online 6 June 2025

results are used as the ground truth to train the solutions in subsequent segments. Consequently, since the predictions in initial segments inevitably contain inaccuracies, these errors propagate and amplify throughout the training process for later segments.

To address this issue, we propose a new training method called DP-PINN. In DP-PINN, the training is divided into two distinct phases. The division is informed by our findings of a pivotal time point, denoted by t_s; the prediction errors became notably more severe after t_s within the time domain $[t_{start}, t_{end}]$, where t_{start} is usually 0.

In phase 1, DP-PINN focuses on training the network to predict solutions from t_{start} (i.e., 0) to t_s. In phase 2, we diverge from the bc-PINN's practice of using the entire solutions predicted within $[0, t_s]$ as the ground truth for subsequent training during $(t_s, t_{end}]$, Rather, we found that what is more important is the accuracy of the solutions predicted at t_s. In phase 1 of DP-PINN, we will obtain the predicted solutions on t_s.

In phase 2, we extend the training across the entire time domain $[0, t_{end}]$ using the same neural network architecture. This phase not only trains the network for $(t_s, t_{end}]$, but also continues to refine the solutions in $[0, t_s]$. Crucially, the solutions predicted at t_s serve as "intermediate" conditions (augmenting the original boundary and initial conditions of the PDE) to guide the training in phase 2. It is critical to determine the appropriate value of t_s. In this paper, we propose two approaches to determining t_s: empirical approach and automated approach.

To further improve the accuracy of predictions at t_s and overall model accuracy, we also propose the new sampling strategies in DP-PINN. With the use of intermediate conditions and the new sampling strategies, DP-PINN is able to achieve much higher accuracy than the state-of-the-art methods.

In summary, DP-PINN incorporates a set of optimization strategies to enhance its prediction accuracy, particularly in solving complex PDEs such as Allen–Cahn equation. These strategies include: (i) strategically dividing the network training into two phases at a specifically identified time point t_s, (ii) identifying an optimized value of t_s automatically, (iii) leveraging the predictions at t_s obtained in phase 1 as the extra "intermediate" conditions to improve accuracy in subsequent predictions, (iv) introducing a learnable parameter in the PINN to improve the accuracy of predicted solutions at t_s obtained in phase 1, and (v) incorporating new sampling strategies to improve the accuracy of solutions at t_s. Together, these strategies empower DP-PINN to effectively solve a variety of PDEs, including "stiff" PDEs.

We evaluated DP-PINN+ with 1D Burger's Equation, 1D Allen–Cahn Equation, 2D and 3D Navier–Stokes Equations (i.e., 2D cylinder wake and 3D unsteady Beltrami flow). We compared DP-PINN+ with the state-of-the-art PINNs in literature, including Time Adaptive PINN, SA-PINN, bc-PINN and NSFNets. The results demonstrate that DP-PINN+ consistently achieves higher prediction accuracy across all tested PDEs, highlighting its improved generalization and robustness.

2. Related work

The standard PINN algorithm can be unstable during training and produce inaccurate approximations around sharp space and time transitions or fail to converge entirely in the solution of "stiff" PDEs, such as the Allen–Cahn equation. Much of the recent studies on PINNs has been devoted to mitigating these issues by introducing modifications to the standard PINN algorithm that can increase training stability and accuracy of the approximation, mostly via splitting the solution domain evenly into several smaller time segments, or by using a weighted loss function during training. We discuss the main approaches of those below.

Non-Adaptive Weighting. The work in [11] points out that the neural network should be forced to satisfy the initial condition closely. Accordingly, a loss function with the form, $\mathcal{L}(\theta) = \mathcal{L}_b(\theta) + C\mathcal{L}_I(\theta) + \mathcal{L}_r(\theta)$, was proposed, where C ($C \gg 1$) is a hyper-parameter.

Adaptive weighting. In [12], PINNs are trained adaptively, using the fully-trainable weights that force the neural network to focus on the difficult regions of the solution, which is an approach reminiscent of soft multiplicative attention masks used in Computer vision [14,15]. The key idea is to increase the weights as the corresponding losses increase, which is accomplished by training the network to minimize the losses while maximizing the weights.

Backward compatible PINN. In [10], the proposed method, termed backward compatible PINN (bc-PINN), addresses the limitation of retraining a single neural network over successive time segments by ensuring that the network satisfies the solutions obtained in all previous time segments (i.e., treating the solutions in previous time segments as the ground truth for the training in subsequent time segments) during the progressive solution of a PDE system. The bc-PINN divides the time axis into several even time intervals, ensuring a comprehensive and backward-compatible solution across the entire temporal range.

Time-Adaptive Approaches. In [11], the time axis is divided into several time intervals, then PINNs are trained on them, either separately or sequentially. The initial condition for each time interval relies on the predictions in the preceding time step. This approach is time consuming due to the dependency between time steps and the need for training multiple PINNs.

NSFnets. As one of the state-of-the-art approaches for solving challenging 2D and 3D Navier–Stokes equations using PINN, NSFnets, proposed in [16], enhances the accuracy and stability of complex flow simulations by deeply integrating fluid physics constraints (such as incompressibility) into the network architecture and training.

Curriculum learning. Curriculum Learning is a training strategy inspired by how humans and animals learn, starting with easier concepts and gradually progressing to more complex ones [17]. In machine learning, this translates to presenting training samples to the model in a curated order, typically beginning with examples that are easier to understand or classify, and then progressively introducing more challenging ones. This approach aims to improve the final model's generalization ability and training speed by guiding the optimization process away from poor local minima and potentially accelerating convergence. The training strategy proposed in this paper bears similarities with the curriculum learning approach. It first identifies and train the phase in the time domain that is relatively easier to generate accurate solutions, and then expands the training to the next phase.

3. Method

In this section, we first give a brief overview of PINN. Next, we present DP-PINN. Finally, we describe the sampling method used in DP-PINN.

3.1. Overview of standard PINN and motivation of DP-PINN

3.1.1. Overview of standard PINN

The general form of the PDE solved by PINN can be defined as follows:

$$u_t + \mathcal{N}[u] = 0, \quad \boldsymbol{x} \in \Omega, t \in [0, t_{end}], \tag{1}$$

$$u(x,t) = g(x,t), \quad \boldsymbol{x} \in \partial\Omega, t \in [0, t_{end}], \tag{2}$$

$$u(x,0) = h(x), \quad \boldsymbol{x} \in \Omega \tag{3}$$

where x is a spatial vector variable, which includes a vector of spatial points on which we need to find solutions of u, t is time, u_t is the partial derivative of u over t, Ω is a subset of R^d (d is the dimension of the space), $\partial\Omega$ denotes the set of all boundary spatial vector variables where the boundary conditions of the PDE are enforced, and $\mathcal{N}[\cdot]$ is non-linear differential operator. Note that Eqs. (2) and (3) represent the boundary conditions and initial conditions of the PDE, respectively.

The solution $u(\boldsymbol{x}, t)$ is approximated by the output $u_\theta(\boldsymbol{x}, t)$ of the deep neural network (θ denotes the network parameters) with inputs $\boldsymbol{x}$ and t. The residual, $r_\theta(x, t)$, is defined as:

$$r_\theta(\boldsymbol{x}, t) = \frac{\partial}{\partial t} u_\theta(\boldsymbol{x}, t) + \mathcal{N}[u_\theta(\boldsymbol{x}, t)], \tag{4}$$

where all partial derivatives can be computed by automatic differentiation methods [18]. With the use of back-propagation [19] during training, the parameters θ can be obtained by minimizing the following loss function:

$$\mathcal{L}(\theta) = \mathcal{L}_b(\theta) + \mathcal{L}_I(\theta) + \mathcal{L}_r(\theta) \tag{5}$$

where $\mathcal{L}_b$ is the loss corresponding to the boundary condition, $\mathcal{L}_I$ is the loss due to the initial condition, and $\mathcal{L}_r$ is the loss corresponding to the residual. $\mathcal{L}_b, \mathcal{L}_I, \mathcal{L}_r$ are essentially penalties for outputs that do not satisfy (2), (3) and (4) respectively. $\mathcal{L}_b$, $\mathcal{L}_I$ and $\mathcal{L}_r$ can be defined by Eqs. (6)–(8), respectively.

$$\mathcal{L}_r(\theta) = \frac{1}{N_r} \sum_{i=1}^{N_r} |r_\theta(x_r^i, t_r^i)|^2 \tag{6}$$

$$\mathcal{L}_b(\theta) = \frac{1}{N_b} \sum_{i=1}^{N_b} |[u_\theta(x_b^i, t_b^i) - g(x_b^i, t_b^i)]|^2, \tag{7}$$

$$\mathcal{L}_I(\theta) = \frac{1}{N_I} \sum_{i=1}^{N_I} |u_\theta(x_I^i, 0) - h(x_I^i)|^2, \tag{8}$$

where $\{x_I^i\}_{i=1}^{N_I}$ and $\{h(x_I^i)\}_{i=1}^{N_I}$ denote the initial points of the PDE and their corresponding values; $\{x_b^i, t_b^i\}_{i=1}^{N_b}$ and $\{g(x_b^i, t_b^i)\}_{i=1}^{N_b}$ are the points on the boundary of the PDE and their values; $\{x_r^i, t_r^i\}_{i=1}^{N_r}$ is the set of collocation points randomly sampled from the domain Ω; N_b, N_I and N_r denote the number of boundary points, initial points and the collocation points, respectively. To tune the parameters θ of the neural network, the training is done for 10k iterations of Adam, followed by 10k iterations of L-BFGS, consistent with the related work for a fair comparison in the experiments. L-BFGS uses Hessian matrix (second derivative) to identify the direction of steepest descent.

3.1.2. Motivation of DP-PINN

We conducted the experiment to use bc-PINN to train the model for solving Allen–Cahn (AC) equation, known for its stiff nature. Fig. 1 visualizes the distribution of its prediction errors (L2-error) across the domain. It can be observed that prediction errors escalate as the training progresses over time. This outcome can be attributed to bc-PINN's unique training approach. bc-PINN divided the entire time domain into four discrete segments, and train the model sequentially across these segments. This scheme allows bc-PINN to focus on smaller, more manageable portions of the time domain at any given moment, which effectively circumvents the challenges faced by the standard PINN that train across the full domain simultaneously. Initially, prediction errors within the early segments may appear harmless. Nonetheless, as the training adopts the outcomes of preceding segments as the ground truth for training in subsequent ones, prediction errors will accumulate as more segments are processed.

The benchmark experiments with bc-PINN provided insights that led us to propose the ideas in our DP-PINN. A key observation is that bc-PINN tends to generate more accurate predictions in the initial segments of the time domain. This can be attributed to two main factors. Firstly, by focusing on a smaller time segment, as opposed to tackling the entire time domain in a single sweep like standard PINN, the model is better positioned to learn the features and relationships among points. This reduced time span simplifies the learning process, enabling more precise predictions. Secondly, the proximity of the early segment to the initial conditions, which are the true ground truth, allows the model to more accurately capture the genuine relationships between points. As the training progresses to points further from the initial conditions, the reliance on previous predictions introduces a compounding effect of inaccuracies. This observation underpins the rationale of the dual-phase training adopted in DP-PINN, aiming to mitigate the propagation of errors.

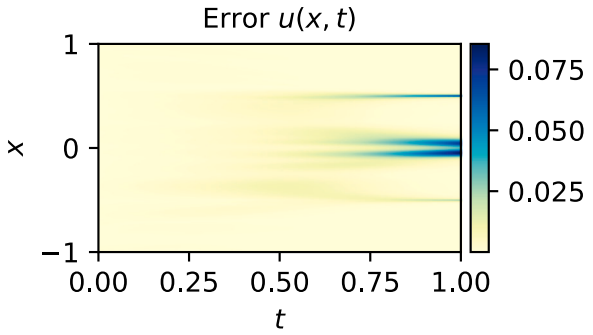

Fig. 1. bc-PINN's absolute prediction error on Allen–Cahn equation.

The core principle behind our DP-PINN involves identifying a critical time point, t_s, beyond which prediction errors notably intensify. We divide the model training into two phases. In phase 1, DP-PINN aims to generate the predictions at t_s as accurate as possible. Subsequently, in the phase-2 training, the predictions at t_s act as the initial condition for training the model over the subsequent time segment $(t_s, t_{end}]$. We anticipate an enhancement in prediction accuracy for $(t_s, t_{end}]$ according to the understanding gleaned from bc-PINN, where predictions are more accurate in segments closer to the initial conditions. We will present the dual-phase training scheme in Section 3.2.

In light of the core principle underpinning DP-PINN, an important objective is to achieve utmost accuracy in predictions specifically at t_s. The accuracy of these predictions at t_s first depends critically on the selection of t_s within the time domain. Identifying a optimal t_s presents a challenge: it must be neither too close to the initial condition at time 0 nor excessively distant. The rationale behind this balance is that because the solutions predicted at t_s will be used as the initial condition for the training over $(t_s, t_{end}]$, smaller t_s means that some points in $(t_s, t_{end}]$ are further away from their initial conditions and may consequently yield more inaccurate predictions.

In this paper, we propose two methods for determining an appropriate value of t_s: an empirical approach and an automated approach.

In the empirical approach, we first conducted benchmark experiments to identify trends in prediction errors of the standard PINN across the time domain, establishing a time range where errors become noticeably larger. Within this range, we experimented with different t_s values to train the DP-PINN+, selecting the t_s that generates the lowest prediction errors. This method requires manual adjustments of t_s and multiple training sessions for DP-PINN+ with different t_s values, which can be both tedious and time consuming, especially when the fine adjustment of t_s is necessary.

Additionally, we have developed a novel automated approach that determines an optimized t_s value with much lower time overhead. The cornerstone of this approach is the identification of a parameter that correlates with the difficulty PINN faces in generating accurate predictions. This parameter is then used to determine the optimized t_s value efficiently.

Determining the value of t_s is part of the phase-1 training of DP-PINN+. In the phase-2 training, we incorporate a trainable parameter η in the model to fine-tune the prediction values at t_s. Determining an optimized value of t_s followed by fine-tuning with η is instrumental in optimizing the accuracy of DP-PINN's predictions and enhancing the model's overall effectiveness.

Another strategy of generating accurate predictions at t_s stems from another observation we made in the benchmark experiments. We applied the standard PINN to train the model with different numbers of sampling points, as shown in Table 1. These experiments revealed a

Table 1
Training accuracy of the standard PINN on the 1D viscous Burger's equation with different numbers of sampling collocation points. The numbers of initial points and boundary points were kept unchanged, which are $N_i = 100$ and $N_b = 50$ respectively, in the three sets of sampling points.

	Number of sampling points		
	5k	10k	20k
Relative L2 error	3.391e−3	**1.375e−3**	2.297e−3

relationship between the number of sampling points and model accuracy: increasing the sampling density generally led to more accurate training results. However, too many sampling points resulted in a decline in model performance, likely due to overfitting. The reason for this trend may be because while a sparse distribution of sampling points challenges the model's ability to learn underlying relationships, excessively dense sampling may cause the model to memorize the training data too closely, losing its generalization capability.

Building on this understanding, we tailor the sampling strategy for DP-PINN to circumvent these issues. Unlike bc-PINN, which samples the same number of points in each time segment, DP-PINN adopts a differentiated approach. Specifically, we allocate a higher density of sampling points to the phase-1 training. This targeted increase in sampling density for phase-1 training is strategic, aimed at obtaining highly accurate predictions at t_s. We will present our sampling strategies in Section 3.3.

3.2. DP-PINN

3.2.1. The dual-phase training scheme

Based on the discussions in Section 3.1.2, we propose a dual-phase training scheme for PINNs. In phase 1, the model is trained on the sampled points in the time duration $[0, t_s]$, and predicts the solutions at t_s (the methods for determining t_s will be presented in next subsection). In phase 2, the same model undergoes training across the entire domain, integrating the predictions at t_s as the intermediate condition, which also acts as the initial condition for the training in $(t_s, t_{end}]$).

The phase-1 training in DP-PINN takes as input the initial conditions of the PDE, the boundary conditions of the points that fall in $[0, t_s]$, and trains the model on the collocation points in $[0, t_s]$. The loss function used by the phase-1 training is defined in Eq. (9), where $\mathcal{L}_I(\theta)$ is the one defined in Eq. (8), representing the initial conditions; $\mathcal{L}_{b1}(\theta)$ represents the compliance to the boundary conditions of the points in the phase-1 domain, which is defined in Eq. (10); $\mathcal{L}_{r1}(\theta)$ represents the residual of the predicted solution on the collocation points in the phase-1 domain, which is defined in Eq. (11); C is a hyper-parameter described in [11], which acts as a weight of the $\mathcal{L}_I(\theta)$ term in the overall loss function.

$$\mathcal{L}_1(\theta) = \mathcal{L}_{b1}(\theta) + C\mathcal{L}_I(\theta) + \mathcal{L}_{r1}(\theta) \tag{9}$$

$$\mathcal{L}_{b1}(\theta) = \frac{1}{N_{b1}} \sum_{i=1}^{N_{b1}} |u_\theta(x_{b1}^i, t_{b1}^i) - g(x_{b1}^i, t_{b1}^i)|^2, \tag{10}$$

$$\mathcal{L}_{r1}(\theta) = \frac{1}{N_{r1}} \sum_{i=1}^{N_{r1}} |r_\theta(x_{r1}^i, t_{r1}^i)|^2, \tag{11}$$

In the phase-2 training, the model is trained across the entire domain $[0, t_{end}]$. Therefore, the original boundary conditions, initial conditions of the PDE are used as the input of the phase-2 training. This phase extends the evaluation of the model's residual to include all sampled collocation points within the entire domain. Moreover, the solutions at time t_s predicted in the phase-1 training are used as an additional condition in the phase-2 training. Based on these, the loss function for the phase-2 training is defined in Eq. (33), where $\mathcal{L}_b(\theta)$ is the one defined in Eq. (7), representing the original boundary conditions of the PDE; $\mathcal{L}_{t_s}(\theta, \boldsymbol{\eta})$ represents the additional intermediate condition established based on the solutions predicted in the phase-1 training at time t_s.

The term $\mathcal{L}_{t_s}(\theta, \boldsymbol{\eta})$ is defined in Eq. (13), where $u'(x_{t_s}, t_{t_s})$ are the values predicted in phase 1 for the collocation points at t_s, and $\boldsymbol{\eta} = (\eta^1, \ldots, \eta^{N_{t_s}})$ is a set of trainable parameters aimed at fine-tuning the predicted values for the N_{t_s} collocation points (i.e., $x_{t_s}^i$) at t_s. $\mathcal{L}_{t_s}(\theta, \boldsymbol{\eta})$ essentially calculates the residual of the predictions at t_s in phase 2 by treating the phase-1 predictions at t_s as "prior". The parameter $\boldsymbol{\eta}$ is updated using Eq. (14) with k indicating the learning step and μ_k the step-specific learning rate.

$$\mathcal{L}_2(\theta, \boldsymbol{\eta}) = \mathcal{L}_b(\theta) + C\mathcal{L}_I(\theta) + \mathcal{L}_r(\theta) + \mathcal{L}_{t_s}(\theta, \boldsymbol{\eta}) \tag{12}$$

$$\mathcal{L}_{t_s}(\theta, \boldsymbol{\eta}) = \frac{1}{N_{t_s}} \sum_{i=1}^{N_{t_s}} |u_\theta(x_{t_s}^i, t_s) - \eta^i * u'(x_{t_s}^i, t_s)|^2 \tag{13}$$

$$\boldsymbol{\eta}(k+1) = \boldsymbol{\eta}(k) - \mu_k \nabla_{\boldsymbol{\eta}} \mathcal{L}_2(\theta, \boldsymbol{\eta}) \tag{14}$$

3.2.2. Determining t_s

In this paper, we propose two approaches, empirical and automated, to determining an optimized value of t_s.

Empirical Approach. In this approach, we first conducted benchmark experiments to observe the time range when the prediction errors of the standard PINN become noticeable. We then select a time point within this range as the initial value of t_s to train DP-PINN+, and record the prediction errors. For example, according to our experimental records, which are also illustrated in Fig. 1, prediction errors start becoming noticeable from around $t = 0.3$ for a total time span of 1.

Subsequently, we manually adjust t_s around this initial value and train DP-PINN with these adjustments until we identify a t_s that results in the minimal prediction errors. While this approach eventually identifies an optimized t_s, it requires extensive manual adjustment and repeated training, which can be both tedious and time-consuming, especially when fine-tuning t_s in small steps over a wide time range.

Automated Approach. Addressing the limitations of the empirical approach, we develop a novel automated method that determines an optimized t_s automatically. This approach leverages insights derived from analyzing the relationship between solution characteristics and prediction difficulty. Initial findings, based on the 1D Allen–Cahn equation, revealed that abrupt changes in solution gradients across the space domain, which represent the "stiffness" of the PDE, tend to lower prediction accuracy for standard PINNs. To illustrate this, Fig. 4 (top plots) depicts standard PINN predictions for Allen–Cahn over the entire time domain, showing less accurate results where gradients change sharply compared to ground truth. Such changes in solution gradients can be characterized by the second derivative of the solution with respect to spatial coordinates (e.g., $\frac{\partial^2 u}{\partial x^2}$ in 1D).

We also extend this analysis to higher-dimensional problems like the 2D/3D Navier–Stokes (NS) equations, which involve multiple output variables (e.g., the velocity components in each dimension and pressure). For each output variable, we compute its second partial derivatives with respect to all spatial input coordinates at each time step. For instance, using a 2D NS equation as an example, for the horizontal velocity component u, this involves calculating $\frac{\partial^2 u}{\partial x^2}$, $\frac{\partial^2 u}{\partial y^2}$, and the mixed partial derivative $\frac{\partial^2 u}{\partial x \partial y}$; this process is then repeated for the other output variables, such as the vertical velocity component v and the pressure p. To capture the overall magnitude of gradient variability irrespective of direction, we take the absolute values of all these second derivatives. Then these values are normalized before being combined into a single, unified distribution for each time step. This final distribution represents the collective intensity of solution gradient changes across all spatial points, all output variables, and all spatial directions at that specific moment in time.

Further analysis of this unified distribution's skewness across time reveals a more detailed picture than initially observed with the 1D

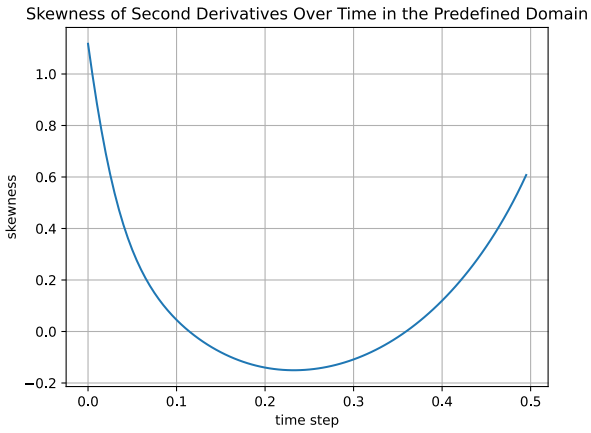

Fig. 2. The skewness of the second derivatives over time within the predefined domain bounded by t_{pre}.

Allen–Cahn equation. While the general concept that skewness reflects prediction difficulty holds, the optimal criterion for selecting t_s depends on the temporal behavior of the skewness value itself, particularly whether it changes sign. The skewness trends can be classified into the following two cases, with t_s determined for each.

Case 1: Skewness changes sign over time. If, within the evaluated time interval $[0, t_{pre}]$, the calculated skewness of the second derivative distribution transitions between positive and negative values (or vice-versa), it signifies a fundamental shift in the asymmetry of gradient changes across the spatial domain (i.e., the distribution's central tendency shifts markedly from one side to the other). This period often corresponds to particularly drastic changes in the system's spatial state. Fig. 2 illustrates this scenario using the Allen–Cahn equation, showing the skewness value crossing zero during the interval. In this scenario, the most challenging point for the PINN is associated with the distribution being closest to symmetric during this transition. Therefore, we select t_s as the time point at which the skewness transitions from decreasing to increasing, which, in this case, corresponds to its minimum within this interval.

Case 2: Skewness remains consistently positive (or negative) over time. If the skewness maintains the same sign throughout the interval $[0, t_{\text{pre}}]$, a different interpretation applies. For positive skewness, a larger value indicates a longer or heavier tail on the right side of the distribution, implying more extreme gradient changes spatially. Fig. 3 illustrate this case for the 2D Navier–Stokes cylinder wake and 3D Navier–Stokes Beltrami flow problems, respectively, where the skewness remains positive. Therefore, in this scenario, we select t_s as the time point at which the skewness ceases to increase. For positive skewness, this signifies that the right tail of the distribution stops growing longer or heavier, implying that the generation of new, extreme positive gradient changes diminishes.

Based on these insights, the automated approach for determining the optimized value of t_s is outlined in Algorithm 1. The algorithm proceeds as follows: (i) An initial time point t_{pre}, different from the starting point 0, is determined (Line 8). (ii) A standard PINN is trained to solve the target PDE over the interval $[0, t_{pre}]$, capturing the solution at every spatial location and each discrete time point within this range (Lines 10–11). (iii) At each discrete time point t in $[0, t_{pre}]$, the second spatial derivative of the solution (e.g., $\frac{\partial^2 u}{\partial x^2}$) is computed across the spatial domain (Lines 1–4). (iv) The skewness of the distribution of $\frac{\partial^2 u}{\partial x^2}$ is calculated for each time step (Line 5). (v) If the skewness consistently changes value in the same direction (monotonically increasing or decreasing) over the time steps up to the current t_{pre}, t_{pre} is advanced to the next discrete time step (lines 13–17). (vi) For this new t_{pre}, the standard PINN prediction is continued to obtain the solution and its corresponding skewness. (vii) Step (v) is repeated until the direction of the skewness change reverses, indicating a trend reversal; the specific value of t_{pre} at which this reversal first occurs is chosen as the optimized t_s(Lines 18–20).

Algorithm 1 Adaptive Determination of t_s

Require: Maximum time to consider T_{max}, Time step Δt
Ensure: Optimal time point t_s
1: **procedure** ComputeSkewnessAtTime(t)
2: Predict solution $u(t)$ at time t using the standard PINN
3: Compute $\mathcal{D}_t = \{\frac{\partial^2 \phi}{\partial x_\alpha \partial x_\beta}(t) \mid \phi \in \{u, v, w, p\},\ \alpha, \beta \in \{x, y, z\}\}$
4: $D_t \leftarrow |\mathcal{D}_t|$
5: **return** Skewness(D_t)
6: **end procedure**

7: Initialize $S \leftarrow []$
8: Initialize $t_{curr} \leftarrow \Delta t$
9: **while** $t_{curr} < T_{max}$ **do**
10: train PINN over domain $[0, t_{curr}]$
11: $s_{curr} \leftarrow$ ComputeSkewnessAtTime(t_{curr})
12: Append (t_{curr}, s_{curr}) to S
13: **if** size(S) ≥ 2 **then**
14: **if** all skewness in S changes in the same direction **then**
15: $t_{curr} \leftarrow t_{curr} + \Delta t$
16: **continue**
17: **end if**
18: **else**
19: $t_s \leftarrow t_{curr}$
20: **Return**
21: **end if**
22: $t_{curr} \leftarrow t_{curr} + \Delta t$
23: **end while**
24: **return** t_s

The rationale behind the strategy for determining t_s is as follows. We use the standard PINN to predict solutions within the time domain $[0, t_{pre}]$. Since t_{pre} begins near the initial time point (i.e., close to the initial condition), it allows for accurate predictions. The bottom four plots in Fig. 4 depict the solutions predicted by the standard PINN over only the first half of the time domain. These predictions are markedly more precise than those shown in the top four plots of the figure. Moreover, Fig. 5 shows the performance errors of the standard PINN.

Given the fairly accurate predictions in $[0, t_{pre}]$, the most beneficial strategy is to identify the time point at which accurate predictions are most challenging (namely the time point t_s where the skewness reverses its trend) and fine-tune the predictions at this time point through the trainable parameters η.

Adopting this approach eliminates the need for multiple trials with different t_s values for training DP-PINN+. Instead, we only require a single set of predictions from the standard PINN within the interval from 0 to t_{pre}. These results are then analyzed to determine the distribution and skewness of the second derivative of the solutions across the spatial domain at each time point. The skewness measurement is finally used to determine the optimized value of t_s.

3.3. Sampling methods

We propose two sampling strategies to provide the training points for DP-PINN. N_r and N_b denote the set of collocation and boundary points sampled in the entire domain, respectively. N_{r1} and N_{b1} denote the set of collocation and boundary points for the phase-1 training, respectively. $N_{r1} \subset N_r$ and $N_{b1} \subset N_b$. N_I denotes the set of initial points. N_{t_s} denotes the set of collocation points sampled at time t_s. In both sampling strategies proposed in this work, we comply with the constraint that the total numbers of sampled collocation points and

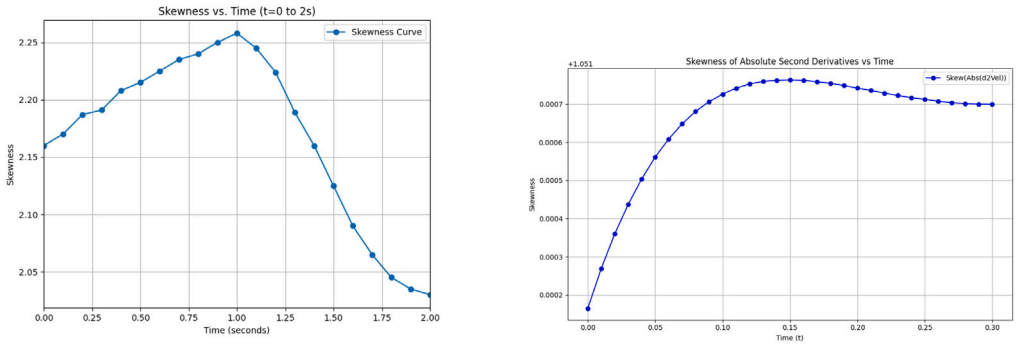

Fig. 3. Skewness over time for the 2D Navier–Stokes cylinder wake (left) and 3D Navier–Stokes Beltrami flow (right).

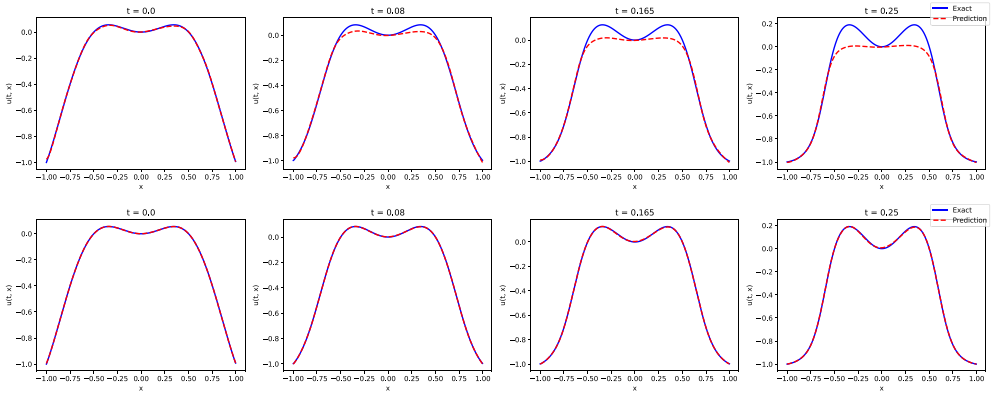

Fig. 4. The predicted solutions of standard PINN at 4 different time steps. (Top) PINN is trained on the entire domain; (Bottom) PINN is trained on the first half of the domain.

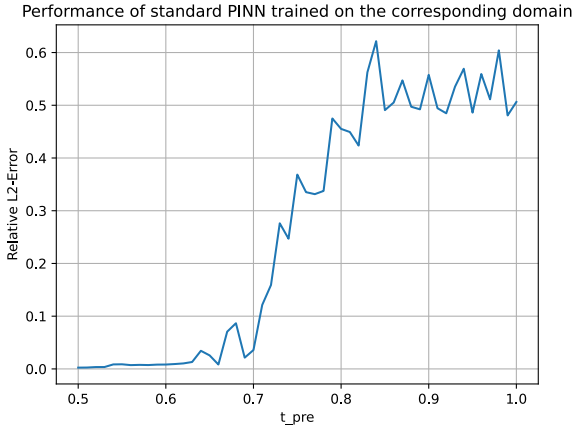

Fig. 5. Prediction errors of the standard PINN for the Allen–Cahn Equation over the time domain $[0,\ t_{pre}]$ as t_{pre} increases.

boundary points that are input for training are no more than $|N_r|$ and $|N_b|$, respectively.

Fixed sampling strategy Unlike [10,11], where the time axis is evenly split into multiple time intervals (e.g., bc-PINN uses 4 intervals), our approach contains two time intervals $[0, t_s]$ and $(t_s, t_{end}]$. In the phase-1 training, i.e., the training in $[0, t_s]$, the number of sampled collocation points is determined by Eq. (15), where $\frac{t_s}{t_{end}} \times |N_r|$ is the number of collocation points that is proportional to the ratio of t_s to t_{end}, and therefore α can be regarded as an amplification factor that increases the sampling density in phase-1. α is a hyper-parameter in training.

$$|N_{r1}| = \alpha \times \frac{t_s}{t_{end}} \times |N_r| \tag{15}$$

Similarly, the number of sampled boundary points is determined by Eq. (16).

$$|N_{b1}| = \alpha \times \frac{t_s}{t_{end}} \times |N_b| \tag{16}$$

In the phase-2 training, the model is trained using the points sampled from both the initial intervals $[0, t_s]$ and the subsequent interval $(t_s, t_{end}]$. In the fixed sampling strategy, the set of sampled points in $[0, t_s]$ in the phase-2 training remains the same as that in the phase-1 training (i.e., N_{r1} and N_{b1}). To adhere to the constraint that the total number of collocation points and boundary points used for training does not exceed $|N_r|$ and $|N_b|$, respectively, we sample $|N_r| - |N_{r1}|$ collocation points and $|N_b| - |N_{b1}|$ boundary points within $(t_s, t_{end}]$ for phase-2 training. Compared to proportional sampling strategy to be presented next, this sampling strategy is deliberately designed to prioritize the accuracy of predictions at the initial stages of the timeline. This focus is particularly important for effectively addressing "stiff" PDEs,such as Allen Cahn equation, where inaccuracies in early predictions can quickly magnify as the model extends to later time points.

Proportional sampling strategy In the former sampling strategy, we preserve the sampling density within $[0, t_s]$ for phase-2 training but sacrifice the sampling density for the subsequent interval $(t_s, t_{end}]$ due to the constraint on the total number of sampling points. In the proportional sampling strategy, after completing phase-1 training, we randomly select a proportion of the total collocation points, $\frac{t_s}{t_{end}} \times |N_r|$, and boundary points, $\frac{t_s}{t_{end}} \times |N_b|$, from the initial sets N_r1 and N_b1 to use during the $[0, t_s]$ interval of phase-2 training. For the remaining time span $(t_s, t_{end}]$, we then allocate the rest of the points, $(1 - \frac{t_s}{t_{end}}) \times |N_r|$ for collocation and $(1 - \frac{t_s}{t_{end}}) \times |N_b|$ for boundary points.

This strategy adjusts the distribution of sampling points in phase-2 training, reducing the number during $[0, t_s]$ so that the number of points in each interval is proportional to their respective duration. Contrary

to fixed sampling, which allocates more sampling points to the early time points, this proportional sampling strategy increases the focus on the later stages (after t_s) in the phase-2 training. This strategy is suited for simpler, non-stiff PDEs.

4. Evaluation

We evaluated DP-PINN+ with 1D Allen–Cahn Equation, 1D Burger's Equation, 2D and 3D Navier–Stokes Equations (i.e., 2D cylinder wake and 3D unsteady Beltrami flow). Allen–Cahn equation, as a "stiff" PDE, is regarded as a most challenging benchmark and commonly used in the literature. 1D Burger's Equation is included in our experiments in order to demonstrate the effectiveness of DP-PINN+ in solving non-stiff PDEs. Furthermore, the evaluation with the 2D and 3D Navier–Stokes equations aims to demonstrate the generalization capability of DP-PINN+ in solving high-dimensional PDEs.

We compared DP-PINN+ with Time Adaptive PINN [11], SA-PINN [12], and bc-PINN [10] on the 1D Allen–Cahn equation, as these represent state-of-the-art PINN approaches for this problem. For the 2D and 3D Navier–Stokes equations, we compared DP-PINN+ with NSFNets [16], which is widely regarded as a state-of-the-art method for solving Navier–Stokes equations.

We carried out the experiments on several PDEs. First, the Allen–Cahn equation, as a "stiff" PDE, is regarded as a most challenging benchmark and was compared against some of the three state-of-the-art PINN algorithms we mentioned. Second, the Burger's equation is relatively easier to solve (non-stiff) and is used as the benchmark in the experiments of SA-PINN; it was included in our experiments in order to demonstrate the effectiveness of DP-PINN in solving non-stiff PDEs. Furthermore, we also conducted experiments on the 2D and 3D Navier–Stokes equations, primarily to demonstrate the generalization capability of DP-PINN in solving high-dimensional problems.

Same as in the literature, we use relative L2-error as the metric to measure the performance of the PINN algorithms. Relative L2-error is defined by Eq. (17), where N_U is the set of sampled points in the entire domain; $u(x,t)$ and $U(x,t)$ represent the predictions and the ground truth at the point x and time t, respectively.

$$Error_{L2} = \frac{\sqrt{\sum_{i=1}^{N_U} |u(x_i,t_i) - U(x_i,t_i)|^2}}{\sqrt{\sum_{i=1}^{N_U} |U(x_i,t_i)|^2}} \tag{17}$$

4.1. Allen–Cahn equation

The Allen–Cahn reaction–diffusion equation is commonly used in the phase-field models, which are often used to model solidification and melting processes, providing insights into the behavior of phase boundaries during these transformations. In this experiment, the Allen–Cahn PDE considered is specified as follows:

$$u_t - 0.0001u_{xx} + 5u^3 - 5u = 0, \quad x \in [-1,1], t \in [0,1], \tag{18}$$

$$u(0,x) = x^2 cos(\pi x), \tag{19}$$

$$u(t,-1) = u(t,1), \tag{20}$$

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

Unlike other PDEs solved by PINNs, Allen–Cahn equation is a nonlinear parabolic PDE that challenges PINNs to approximate solutions with sharp space and time transitions, and also introduces periodic boundary conditions (Eqs. (20)–(21)). High-fidelity data from [6] is used as a reference and for providing boundary and initial data for training purpose.

In order to deal with this periodic boundary conditions, the loss function $\mathcal{L}_b(\theta)$ and $\mathcal{L}_{b1}(\theta)$ defined in Eqs. (7) and (10) are replaced by:

$$\mathcal{L}_b(\theta) = \frac{1}{N_b}\sum_{i=1}^{N_b} |u_\theta(-1,t_b^i) - u_\theta(1,t_b^i)|^2 + |u_{x\theta}(-1,t_b^i) - u_{x\theta}(1,t_b^i)|^2, \tag{22}$$

Table 2
Comparing the performance of different PINN algorithms in solving the Allen–Cahn equation. When testing the PINN algorithms in literature, the settings are exactly same as those reported in the literature whenever applicable.

Methods	Relative L2-Error
Standard PINN	9.92e−1 ± 5.41e−3
Time-adaptive approach	7.55e−2 ± 1.03e−2
SA-PINN	2.06e−2 ± 1.33e−2
bc-PINN	1.67e−2 ± 8.95e−3
DP-PINN (Empirical t_s)	**8.45e−3 ± 2.93e−3**
DP-PINN (Automated t_s)	**8.21e−3 ± 3.03e−3**

$$\mathcal{L}_{b1}(\theta) = \frac{1}{N_{b1}}\sum_{i=1}^{N_{b1}} |u_\theta(-1,t_{b1}^i) - u_\theta(1,t_{b1}^i)|^2 + |u_{x\theta}(-1,t_{b1}^i) - u_{x\theta}(1,t_{b1}^i)|^2, \tag{23}$$

The neural network architecture is fully connected with 4 hidden layers, each with 128 neurons. The input layer takes two inputs (two neurons) - x and t. The output is the prediction of $u_\theta(x,t)$. This architecture is identical to that used in [11,12], which allows a fair comparison. We set the number of collocation, initial and boundary points to $|N_r| = 20\,000$, $|N_I| = 512$ and $|N_b| = 100$. The training is conducted by 10k L-BFGS iterations in phase 1, and 60k iterations of Adam only in phase 2. The amplification factor α as a hyper-parameter is set to 1.6. The number of the points sampled at time t_s for the phase-1 training is set to $|N_{t_s}| = 512$, excluding the points at the boundary. t_s is set to 0.3, and the trainable co-efficient η in Eq. (13) are initialized following a uniform distribution in the range of [0,1). Mini-batches are used in training, with the batch size of 32. All experiments underwent 10 independent runs with random starts and the performance reported is the average over the 10 runs.

Table 2 shows the performance of different PINNs in solving the Allen–Cahn equation. Comparing to other state-of-the-art algorithms, DP-PINN with empirical t_s can achieve a much lower relative L2 error, which is 8.45e−3 ± 2.93e−3. Note that standard PINN cannot solve the Allen–Cahn equation.

Fig. 6 visualizes the numerical solutions and prediction errors obtained by DP-PINN in Table 2. Comparatively, Fig. 1 visualizes the prediction errors obtained by bc-PINN in this experiment. It can be seen from the figures that the predictions made by DP-PINN are very accurate.

To evaluate the effectiveness of DP-PINN+ with automated t_s, we implemented the automated approach described in Section 3.2.2 For the Allen–Cahn equation, the value of t_s that was found by the automated approach is 0.25, resulting in a similar prediction error obtained by DP-PINN with empirical t_s (all other experimental settings are the same). The prediction errors obtained by DP-PINN with automated t_s are visualized in Fig. 7.

4.2. 1D viscous Burger's equation

The 1D viscous Burger's equations with Dirichlet boundary conditions are formalized as follows:

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0, \quad x \in [-1,1], t \in [0,1], \tag{24}$$

$$u(0,x) = -sin(\pi x), \tag{25}$$

$$u(t,-1) = u(t,1) = 0. \tag{26}$$

In this set of experiments, the neural network architecture is fully connected with 8 hidden layers, each with 20 neurons. This architecture is identical to [6,12]. We set $|N_r| = 10\,000$, $|N_b| = 50$ and $|N_I| = 100$ for all PINNs, which is the common setting in [6,12]. High-fidelity data from [6] is used as a reference and for providing boundary and initial data for training purpose. All training is done by 10k Adam iterations, followed by 10k L-BFGS iterations, which is the same as those used in literature. For DP-PINN, $|N_{t_s}| = 256$. η is initialized in the

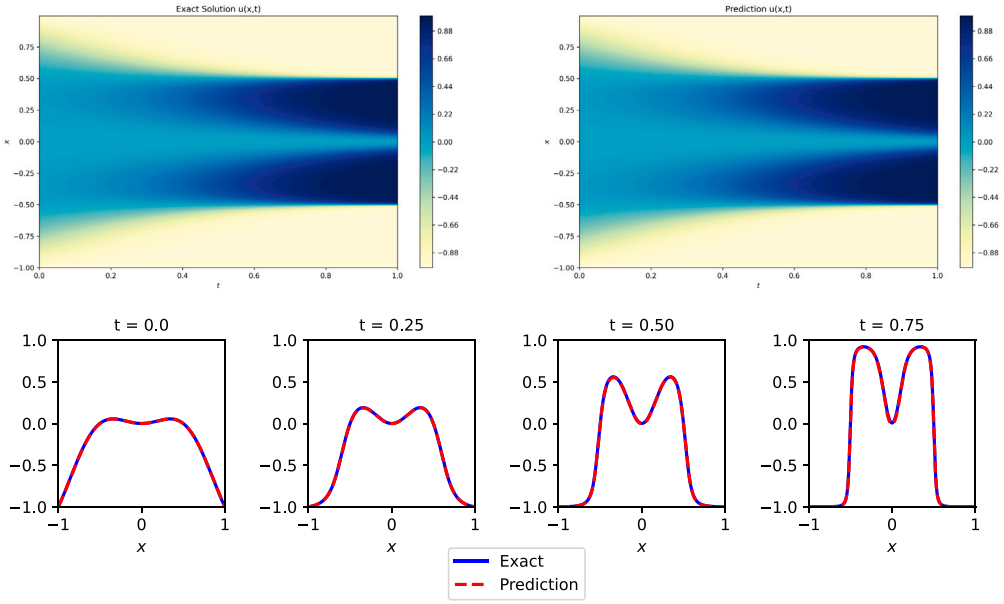

Fig. 6. Visualization of the exact solutions of Allen Cahn equation and the prediction obtained by DP-PINN. The two plots on the top are the exact solution (left) and the prediction (right) across the domain. The four plots at the bottom are the predicted solution at 4 different time point.

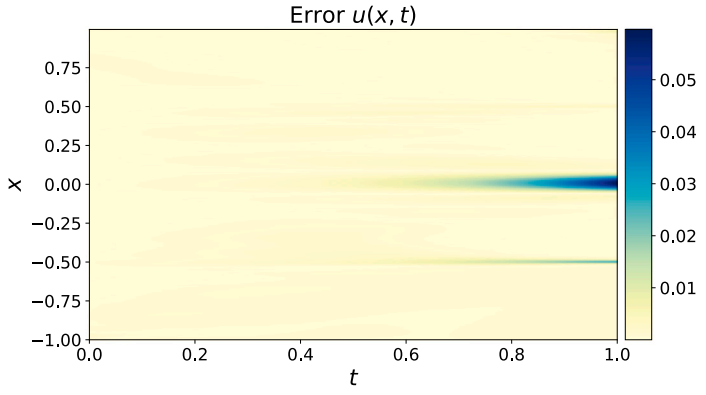

Fig. 7. Visualization of the prediction error distribution for the Allen–Cahn equation with automated t_s.

same way as in the previous experiment. In this experiment, we only compared our DP-PINN with SA-PINN since other two state-of-the-art PINN algorithms (bc-PINN and Time Adaptive) are not tested on this simpler PDE.

The performance of different PINN methods are listed in Table 3. From this table, we can see that DP-PINN achieves slightly better performance than SA-PINN. The improvement is not as prominent as with Allen Cahn equation because Burger's equation is easy to solve. The state-of-the-art PINN method (SA-PINN) can already generate very accurate solutions.

Fig. 8 visualizes the solutions of the Burger's equation and their prediction errors obtained by DP-PINN in Table 3. Once again, the solutions generated by DP-PINN are very accurate.

We also used DP-PINN with automated t_s to predict the solutions to Burger's equation. The prediction error is listed in Table 3, which once again shows that our automated approach to determining t_s is effective.

Table 3
Performance of different PINNs algorithms in solving Burger's equation.

Methods	Relative L2 error
Standard-PINN	1.375e−3 ± 1.191e−3
SA-PINN	4.685e−4 ± 1.211e−4
DP-PINN (Empirical t_s)	**4.595e−4 ± 1.381e−4**
DP-PINN (Automated t_s)	**3.710e−4 ± 0.348e−4**

4.3. 2D cylinder wake

The flow around a two-dimensional cylinder is a canonical problem in fluid dynamics, often used to study flow separation and unsteady vortex shedding phenomena. For an incompressible viscous fluid flowing past a cylinder, the velocity components $u(x, y, t)$, $v(x, y, t)$ and pressure $p(x, y, t)$ are governed by the 2D incompressible Navier–Stokes

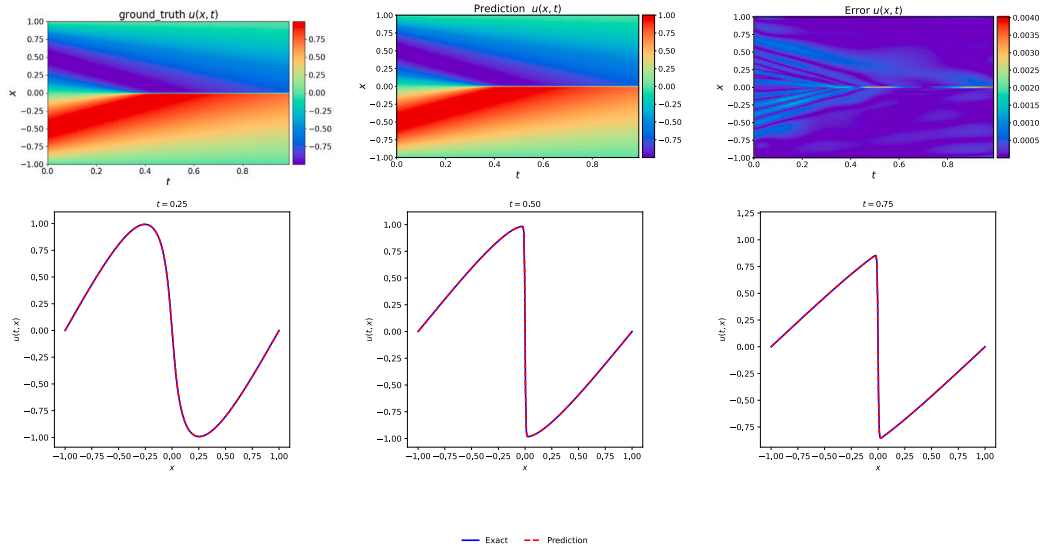

Fig. 8. Visualization of the solution of the 1D viscous Burger's equation and the prediction errors obtained by DP-PINN. The three plots on the top show the exact solution (left) predicted solution (middle) and prediction error (right) across the entire domain. The plots at the bottom show the predicted solutions at three different time points.

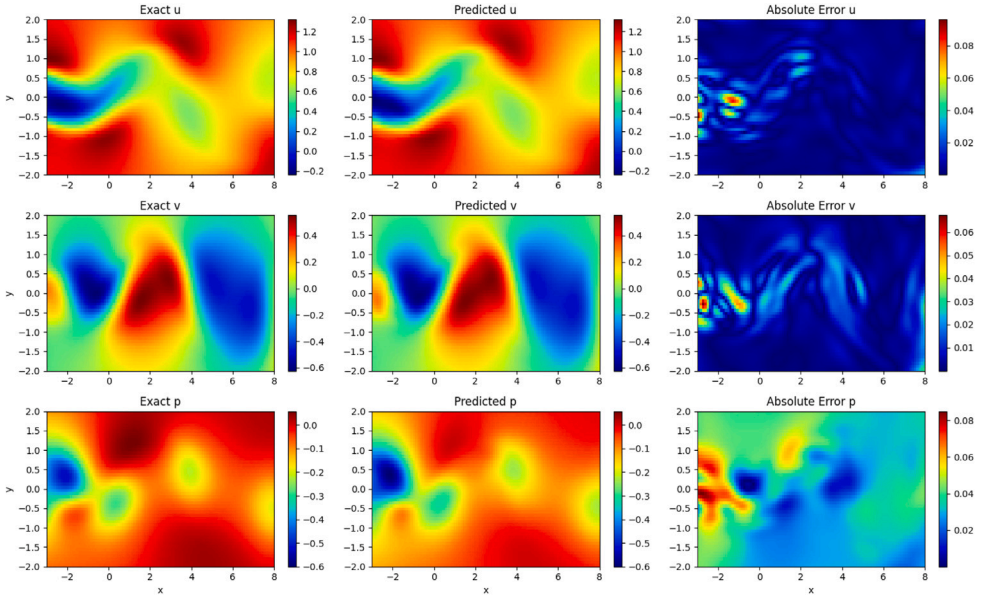

Fig. 9. Ground truth (left column), prediction (middle column) and error distribution (right column) for velocity (first 2 rows) and pressure (third row) field from DP-PINN.

equations in Eqs. (27)–(29).

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \tag{27}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right), \tag{28}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right). \tag{29}$$

where ρ is the fluid density, ν is the kinematic viscosity and $Re = 100$. High-fidelity DNS data from [6] is used as a reference and for providing data points for training and evaluation purpose.

The 2D cylinder wake problem is considered challenging for PINNs due to: (1) vortex shedding, which introduces rapidly varying features in both space and time [20], and (2) multiscale dynamics that involve both slow (bulk flow) and fast (vortical structures) variations [21].

In this experiment, we follow the same setting as in [16] and consider a domain in the range of $[1, 8] \times [-2, 2]$ and the time interval is $[0, 20]$ with time step $\Delta t = 0.1$.

As for the training data, we have chosen a total of $N = 10{,}000$ data points across the domain. The loss function used for this experiment is subsequently defined as follows:

$$\mathcal{L}_1(\theta) = \mathcal{L}_{data}(\theta) + \mathcal{L}_{PDE}(\theta). \tag{30}$$

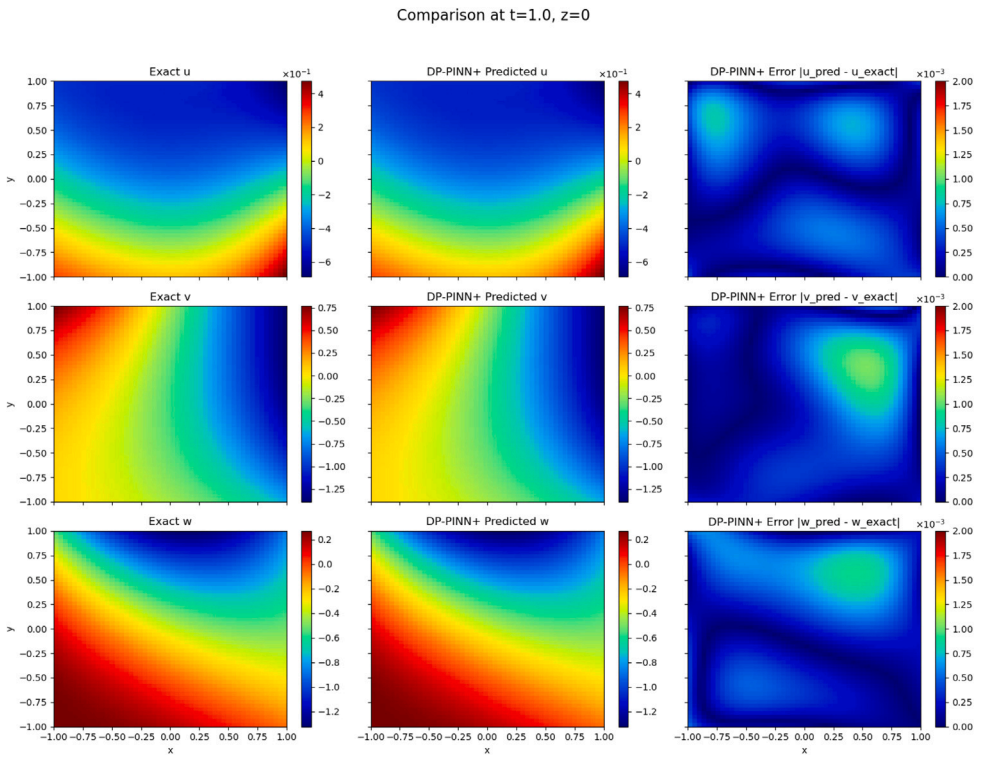

Fig. 10. Prediction and error distribution for DP-PINN with automated t_s selection approach, at $t = 1.0$ on the plane $z = 0$.

$$\mathcal{L}_{data}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[(u_\theta(x_i, y_i, t_i) - u(x_i, y_i, t_i))^2 + (v_\theta(x_i, y_i, t_i) - v(x_i, y_i, t_i))^2\right]. \tag{31}$$

$$\mathcal{L}_{PDE}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[f_u(u_\theta, v_\theta, p_\theta)^2 + f_v(u_\theta, v_\theta, p_\theta)^2 + f_c(u_\theta, v_\theta, p_\theta)^2\right]. \tag{32}$$

Equation of f_u, f_v, f_c were defined in Eqs. (27), (28) and (29). In phase 2, the PINN is trained using the following loss equation:

$$\mathcal{L}_2(\theta, \boldsymbol{\eta}) = \mathcal{L}_{data}(\theta) + \mathcal{L}_{PDE}(\theta) + \mathcal{L}_{t_s}(\theta, \boldsymbol{\eta}) \tag{33}$$

where $\mathcal{L}_{t_s}(\theta, \boldsymbol{\eta})$ is defined in Eq. (13).

The neural network architecture is a fully connected network with 8 hidden layers, each containing 20 neurons. Training is performed using 10k epochs of Adam, followed by 50k iterations of L-BFGS. For DP-PINN+, we set $|N_{ts}| = 5000$.

We compare DP-PINN+ with NSFNets [16], a state-of-the-art PINN for Navier–Stokes Equations, and also with bc-PINN, the best performing PINN for solving Allen–Cahn Equation as shown in Section 4.1. The mean relative L_2 errors are reported in Table 4, and visualizations of the ground truth, predictions of DP-PINN+, and absolute errors are shown in Fig. 9.

Experiment results show that when solving the 2D Navier–Stokes equations, DP-PINN+ achieves higher prediction accuracy compared to NSFnets. Specifically, the L_2 error was reduced from 1.45e−2 to 1.34e−2 for u (x-direction velocity), from 4.18e−2 to 3.46e−2 for v (y-direction velocity), and from 3.30e−1 to 2.84e−1 for pressure p. Additionally, when comparing DP-PINN trained using empirical vs. automated selection of t_s, the automated approach yields lower error. This improvement arises because the empirical approach relies on manual "trial-and-error" and failed to capture the optimal t_s.

Table 4
Mean relative L2 error across all time steps.

Methods	Relative L_2 error		
	(u)	(v)	(p)
NSFnets	1.48e−2	4.18e−2	3.30e−1
bc-PINN	7.89e−2	1.38e−1	1.1256
DP-PINN (Automated t_s)	**1.34e−2**	**3.46e−2**	**2.84e−1**

Table 5
Mean relative L2 error across all time steps.

Methods	Relative L_2 error		
	(u)	(v)	(w)
NSFnets	3.759e−4	2.592e−4	3.389e−4
bc-PINN	1.132e−3	2.004e−3	2.853e−3
DP-PINN (Automated t_s)	**3.369e−4**	**1.986e−4**	**2.807e−4**

4.4. 3D unsteady Beltrami flow

3D Beltrami flow form a class of special solutions to the incompressible Navier–Stokes equations in which the vorticity is parallel to the velocity field. In an incompressible viscous flow, the velocity components $u(x, y, z, t)$, $v(x, y, z, t)$, $w(x, y, z, t)$ and pressure $p(x, y, z, t)$ must satisfy Eqs. (34)–(37).

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0, \tag{34}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right), \tag{35}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2}\right), \tag{36}$$

$$\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} = -\frac{1}{\rho}\frac{\partial p}{\partial z} + \nu\left(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2}\right). \tag{37}$$

where ρ is the fluid density and ν is the kinematic viscosity.

The Beltrami condition requires:

$$\nabla \times \mathbf{u} = \lambda(\mathbf{x}, t)\,\mathbf{u}. \tag{38}$$

where λ is the Beltrami parameter. Data from [22] is used as a reference and for providing data points for training.

3D Beltrami flow is considered challenging for PINNs due to: (1) the high dimensionality of the 3D spatial domain [23], (2) the curl-based constraint, which requires second-order mixed derivatives and increases training complexity and instability [13], (3) sharp gradients that introduce stiffness [13], and (4) multiscale behavior that is difficult for PINNs to capture in a unified way [23].

For the 3D Beltrami flow, we compare DP-PINN+ with NSFNets [16] and bc-PINN, as in the 2D cylinder wake. In this experiment, we adopted the following identical setup to that in NSFNets [16]. The computational domain is defined as $[-1,1] \times [-1,1] \times [-1,1]$, with a time interval of $[0,1]$. The neural network architecture is a fully connected network with 7 hidden layers, each containing 50 neurons. The domain boundaries are discretized into a 31×31 grid per face, and we set $|N_r| = 10{,}000$, $|N_b| = 59{,}400$, and $|N_I| = 29{,}791$ for training.

Training is performed using 30k epochs of Adam with a learning rate of 10^{-3}, followed by 10k iterations of L-BFGS. For DP-PINN+, we choose $|N_{ts}| = 29{,}791$. The mean relative L_2 error is reported in Table 5. A snapshot of the velocity field and its corresponding error distribution at $t = 1.0$ and $z = 0$ is shown in Fig. 10.

In this high-dimensional Beltrami flow problem, the proposed DP-PINN training scheme achieves higher predictive accuracy than NSFNets and BC-PINN, reducing the final relative L_2 error across all velocity components.

5. Conclusion and future works

In this paper, we develop a new training method for PINN, called DP-PINN+, to address the limitations of existing PINNs in solving "stiff" PDEs. By dividing the training into two phases at a carefully chosen time point t_s, DP-PINN+ significantly reduces prediction errors that accumulate over time in existing methods like bc-PINN. The first phase of DP-PINN focuses on training the network and predicting accurate solutions at t_s, while the second phase trains over the entire time domain, using the solutions at t_s as an intermediate condition. We also developed a method to automatically determine an optimized value of t_s based on the skewness of the distribution of the solution's second derivative over the space domain. Experiments were conducted to compare DP-PINN+ with state-of-the-art PINNs, including Time-Adaptive PINN, SA-PINN, bc-PINN and NSFNets. We evaluated the PINN methods with 1D Allen–Cahn Equation, 1D Burger's equation, 2D and 3D Navier–Stokes Equations (i.e., 2D cylinder wake and 3D Beltrami flow). Experimental results show that our DP-PINN+ achieves higher prediction accuracy than the existing PINNs. Future work has been planned. On the one hand, there has not been theoretical analysis regarding the impact of network size on the approximation accuracy of PDE solutions. On the other hand, we plan to explore techniques for solving PDEs from other application domains such as chemistry and biology.

CRediT authorship contribution statement

Da Yan: Writing – original draft, Validation, Methodology. **Ligang He:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] D. Yan, L. He, DP-PINN: A dual-phase training scheme for improving the performance of physics-informed neural networks, in: International Conference on Computational Science, Springer, 2024, pp. 19–32.
[2] W.F. Ames, Numerical Methods for Partial Differential Equations, Academic Press, 2014.
[3] J.Y. Lee, S. Ko, Y. Hong, Finite element operator network for solving parametric pdes, 2023, arXiv preprint arXiv:2308.04690.
[4] M.H. Zawawi, A. Saleha, A. Salwa, N. Hassan, N.M. Zahari, M.Z. Ramli, Z.C. Muda, A review: Fundamentals of computational fluid dynamics (CFD), in: AIP Conference Proceedings, Vol. 2030, AIP Publishing, 2018.
[5] S. Huang, W. Feng, C. Tang, J. Lv, Partial differential equations meet deep neural networks: A survey, 2022, arXiv preprint arXiv:2211.05567.
[6] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[7] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in pytorch, 2017.
[8] M. Dissanayake, N. Phan-Thien, Neural-network-based approximations for solving partial differential equations, Commun. Numer. Methods Eng. 10 (3) (1994) 195–201.
[9] R.L. Burden, Numerical Analysis, Brooks/Cole Cengage Learning, 2011.
[10] R. Mattey, S. Ghosh, A novel sequential method to train physics informed neural networks for allen cahn and cahn hilliard equations, Comput. Methods Appl. Mech. Engrg. 390 (2022) 114474.
[11] C.L.W.C.L. Wight, J.Z.J. Zhao, Solving allen-cahn and cahn-hilliard equations using the adaptive physics informed neural networks, Commun. Comput. Phys. 29 (3) (2021) 930–954.
[12] L. McClenny, U. Braga-Neto, Self-adaptive physics-informed neural networks using a soft attention mechanism, 2020, arXiv preprint arXiv:2009.04544.
[13] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, SIAM J. Sci. Comput. 43 (5) (2021) A3055–A3081.
[14] Y. Pang, J. Xie, M.H. Khan, R.M. Anwer, F.S. Khan, L. Shao, Mask-guided attention network for occluded pedestrian detection, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 4967–4975.
[15] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, X. Tang, Residual attention network for image classification, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 3156–3164.
[16] X. Jin, S. Cai, H. Li, G.E. Karniadakis, NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations, J. Comput. Phys. 426 (2021) 109951.
[17] Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, in: Proceedings of the 26th Annual International Conference on Machine Learning, 2009, pp. 41–48.
[18] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, J. Marchine Learn. Res. 18 (2018) 1–43.
[19] D.E. Rumelhart, R. Durbin, R. Golden, Y. Chauvin, Backpropagation: The basic theory, in: Backpropagation, Psychology Press, 2013, pp. 1–34.
[20] P.K. Kundu, I.M. Cohen, D.R. Dowling, J. Capecelatro, Fluid Mechanics, Elsevier, 2024.
[21] C. Wiliamson, Vortex dynamics in the cylinder wake, 1996.
[22] C.R. Ethier, D. Steinman, Exact fully 3D Navier–Stokes solutions for benchmarking, Internat. J. Numer. Methods Fluids 19 (5) (1994) 369–375.
[23] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nat. Rev. Phys. 3 (6) (2021) 422–440.