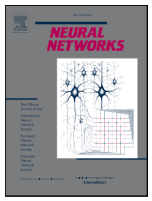
Neural Networks 191 (2025) 107841

Contents lists available at ScienceDirect

Neural Networks

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



Full Length Article

Loss-driven dynamic weight and residual transformation in physics-informed neural network

Yabin Zhang [a], Liang-Jian Deng [b,*]

[a] School of Mathematical Sciences, University of Electronic Science and Technology of China, Chengdu, Sichuan, 611731, China
[b] School of Mathematical Sciences/Multi-Hazard Early Warning Key Laboratory of Sichuan Province, University of Electronic Science and Technology of China, Chengdu, Sichuan, 611731, China

ARTICLE INFO

Keywords:
Dynamic weight
Neural tangent kernel
Multi-parameters discovery
Residual transformation

ABSTRACT

Physics-Informed Neural Networks (PINNs) have attracted substantial interest as a powerful approach for addressing both forward and inverse problems associated with partial differential equations (PDEs). In this study, we reveal a persistent phenomenon of imbalance within the empirical risk loss function. Building upon this observation, we introduce a loss-driven dynamic weight PINN, along with a theoretical analysis grounded in the neural tangent kernel. To assess the efficacy of our proposed method, we conduct comprehensive evaluations across various complex, time-dependent physical phenomena, employing different learning rate decay strategies. A series of experiments are carried out to scrutinize the influence of dynamic weight update frequencies and allocation principles on both the accuracy and computational efficiency. Furthermore, to shed light on the underlying causes of PINN failures in solving inverse problem, we offer a theoretical explanation that involves multi-parameter discovery via gradient descent for high-dimensional nonlinear PDEs. Leveraging the concept of limiting equivalence, we propose an innovative algorithm, termed Residual Transformation PINN (ResTran-PINN), which not only accelerates convergence but also diminishes computational time. These studies provide new insights and a comprehensive framework for understanding PINN optimization and diagnosing possible failure modes.

1. Introduction

In recent years, there has been a surge of interest in applying advanced neural network architectures to complex systems such as traffic flow prediction, resource allocation in edge-cloud computing, and wireless sensor networks in E-commerce (Ali et al., 2025a, 2024, 2025b; Ullah et al., 2024). Currently, deep learning (LeCun et al., 2015) combined with the laws of underlying physics has emerged as a new subfield named Scientific Machine Learning (Baker et al., 2019). A typical method is the Physics-Informed Neural Network (PINN) (Raissi et al., 2019), which has been widely concerned with dealing with the forward and inverse problems of partial differential equations (PDEs). PINN is based on the neural network that collaborated with some laws of physics and initial-boundary value conditions to approximate the solution of PDEs. At the same time, it computes the derivatives of spatio-temporal variables using the automatic differentiation technique (Baydin, 2018). Physical laws, including conservation laws, symmetry, energy, mass, and so on, are used to constrain neural networks to increase the confidence of results, such as conservative PINN (Jagtap et al., 2020), two-stage PINN with conserved quantities (Lin & Chen, 2022), symmetry PINNs (Zhu et al., 2022). In addition, PINN has also been extended to investigate fractional differential equations (Pang et al., 2019), stochastic differential equations (Yang et al., 2020; Zhang et al., 2020, 2019), integro-differential equations (Lu et al., 2021; Yuan et al., 2022), etc.

Despite the PINNs having achieved a series of empirical successes in recent years, there remain complex physical phenomena whose characteristics are challenging to identify accurately. To delve deeper into and scrutinize potential failure modes within PINNs, a comprehensive examination of some pieces of evidence and theoretical analyses has been conducted from diverse perspectives. Ref. (Wang et al., 2021) has illuminated that the failure of PINNs is intricately linked to the unbalanced back-propagation gradient, the numerical stiffness encountered during model training. To tackle this pressing issue, researchers have proposed a learning rate annealing algorithm grounded in gradient statistics over the training process. Notably, the neural tangent kernel (NTK) (Jacot et al., 2018) of PINNs has been introduced to dissect the training dynamics of PINNs from a limiting NTK perspective, revealing distinct convergence patterns among various loss components (Farhani et al., 2025;

* Corresponding author.
E-mail addresses: zhangybccut@126.com (Y. Zhang), liangjian.deng@uestc.edu.cn (L.-J. Deng).

https://doi.org/10.1016/j.neunet.2025.107841
Received 10 March 2025; Received in revised form 2 July 2025; Accepted 6 July 2025
Available online 9 July 2025

Wang et al., 2022). Under suitable conditions, this theoretical framework converges to a deterministic kernel, maintaining its constancy throughout training in an infinitely wide form Wang et al. (2022). The aforementioned strategies effectively bridge the optimization of PINNs with multi-task learning paradigms, wherein the introduction of adaptive weights plays a pivotal role. These adaptive weights are instrumental in alleviating the inherent balance contradiction among the loss components, ensuring a more harmonious and efficient training process. In this study, our objective is to design an efficient algorithm. This algorithm should not only possess a simple structure but also effectively avoid an increase in additional computational costs.

1.1. Related work and our contributions

The weights between the loss components have been identified as a critical factor affecting PINNs' performance (Wang et al., 2021). Note that there have been some algorithms that alleviate the balance conflict between loss components from different perspectives. When simulating multi-scale, chaotic, or turbulent systems, PINNs often encounter convergence difficulties due to their failure to respect the inherent spatio-temporal causal structure of physical systems. To address this deficiency, the study introduces additional temporal weights to respect the causal structure (Wang et al., 2024). SA-PINNs incorporate trainable weights for each training point, a mechanism akin to the soft multiplicative attention masks employed in computer vision (McClenny & Braga-Neto, 2023). Consequently, points with challenging areas of the solution space are automatically assigned greater weights within the loss function. This weighting strategy compels the approximation to achieve enhanced accuracy specifically at those points, but it maybe increase the computing cost (McClenny & Braga-Neto, 2023). A self-adaptive loss weighting strategy was proposed to automatically determine the weights of losses by adjusting adaptive parameters in every epoch, leveraging maximum likelihood estimation (Xiang et al., 2022). This strategy causes the loss at the boundary conditions to fluctuate sharply. Ref. Bischof and Kraus (2025) proposed a relative loss balancing with random lookback method, which endeavors to integrate the most advantageous features of the previously existed methods into a novel framework. This approach also relies on hyperparameters, such as exponential decay rates and random variables. Loss-attentional PINN was introduced to dynamically update weights, which equips an independent loss-attentional network for each loss component (Song et al., 2024). Despite it demonstrates remarkable accuracy and convergence speed, it lacks some in-depth theoretical analysis. Solely focusing on balancing loss magnitudes, most existing adaptive weighting approaches for PINNs neglect the cost of time consumption. These strategies maybe face critical limitations when extended to high-dimensional PDEs.

In this paper, we elucidate the intrinsic connection between the optimization of PINNs and multi-task learning, uncovering the phenomenon of stable imbalance inherent in the empirical risk loss function for solving the high-dimensional equation. Leveraging this insight, we introduce a PINN variant equipped with a loss-driven dynamic weight mechanism. This innovative approach dynamically adjusts the weight assigned to boundary loss component during training without introducing any additional computational overhead, thereby maintaining training efficiency. We further derive its NTK theory and analyze the stability of the associated matrix. To evaluate the efficacy of our proposed algorithm, we integrate various learning rate decay strategies and apply them to several complex physical scenarios governed by high-dimensional nonlinear PDE with high-order terms. These serve as robust numerical examples to verify the performance of our proposed algorithm. A series of experiments is conducted to explore the effects of update frequency and allocation principle for dynamic weight on accuracy and efficiency. In the context of inverse problem, we develop a theoretical framework for multi-parameter discovery using PINNs for high-dimensional nonlinear PDEs, based on gradient descent, to understand why PINNs may occasionally fail. Drawing inspiration from the concept of limiting equivalence, we propose a novel algorithm, the Residual Transformation PINN, designed to accelerate parameter convergence and decrease computational time. Our contributions are outlined as follows:

- **Loss-driven dynamic weight PINN for forward problem**:
  - We identify a persistent but stable imbalance among different loss components in PINNs, which hinders their convergence and generalization.
  - Based on this, we propose a loss-driven dynamic weight PINN method that dynamically adjusts the weight of loss component during training.
  - We theoretically analyze the NTK of the proposed algorithm, demonstrating dynamic weight improves the model's ability to capture complex physical dynamics.
  - Our experiments systematically investigate the impact of update frequency and allocation principles for dynamic weight on training dynamics, providing insights into optimal hyperparameter choices for PINNs.
- **Residual transformation PINN for inverse problem**:
  - We provide a theoretical framework for solving inverse problem using PINNs based on gradient descent, elucidating the interplay between the optimization landscape and the physics-informed loss.
  - Drawing inspiration from the concept of limiting equivalence, we introduce Residual Transformation PINN (ResTranPINN), an algorithm that accelerates convergence by dynamically transforming the residual terms in the loss function.
  - Through extensive experiments, we demonstrate ResTranPINN's superiority in multi-parameter discovery for high-dimensional nonlinear PDEs, both with and without noisy data, highlighting its robustness and practical utility.

1.2. Organization

This paper is structured as follows. In Section 2, we provide a concise overview of the three-layer wide-width fully-connected neural networks and its Neural Tangent Kernel (NTK). Section 3 introduces the high-dimensional Kadomtsev-Petviashvili equation, the Physics-Informed Neural Network (PINN) method, the proposed dynamic weight algorithm, and experimental results for the forward problem. In Section 4, we delve into the neural tangent kernel for the inverse problem of PINN, propose a Residual Transformation PINN method, and present experimental results for the inverse problem. The paper concludes with a summary and discussion.

2. Wide-width neural network

In this section, we first review the definition of an infinite-width neural network and its NTK theory, which will be involved in the given method. Suppose the wide-width neural network model has an input layer, two hidden layers, and an output layer with d_0, d_1, d_2, d_3 neurons per layer, respectively. Let input $\boldsymbol{x} \in \mathbb{R}^{d_0}$ have

$$\begin{aligned} u^{(0)}(\boldsymbol{x}) &= \boldsymbol{x} \in \mathbb{R}^{d_0}, \\ u^{(1)}(\boldsymbol{x}) &= \frac{1}{\sqrt{d_0}} \boldsymbol{W}^{(1)} u^{(0)}(\boldsymbol{x}) + \boldsymbol{b}^{(1)} \in \mathbb{R}^{d_1}, \\ u^{(2)}(\boldsymbol{x}) &= \frac{1}{\sqrt{d_1}} \boldsymbol{W}^{(2)} \sigma\left(u^{(1)}(\boldsymbol{x})\right) + \boldsymbol{b}^{(2)} \in \mathbb{R}^{d_2}, \\ u^{(3)}(\boldsymbol{x}) &= \frac{1}{\sqrt{d_2}} \boldsymbol{W}^{(3)} \sigma\left(u^{(2)}(\boldsymbol{x})\right) + \boldsymbol{b}^{(3)} \in \mathbb{R}^{d_3}, \end{aligned} \tag{1}$$

where the trained parameters are $\boldsymbol{W}^{(i)} \in \mathbb{R}^{d_i \times d_{i-1}}$ and $\boldsymbol{b}^{(i)} \in \mathbb{R}^{d_i}$ $(i = 1, 2, 3)$, and σ is a nonlinear smooth activation function. Additionally, $u^{(i)}(\boldsymbol{x})$ represents the network output at the i-th layer. All trainable parameters are represented as $\theta = \{\boldsymbol{W}^{(i)}, \boldsymbol{b}^{(i)}\}_{i=1}^{3}$, which are initialized to the independent and identically distributed standard normal distribution $\mathcal{N}(0, 1)$. This parameterization is called NTK parameterization

(Jacot et al., 2018). The parameters θ are expressed as θ_t at the time t as the model is trained, and the corresponding output $u_t(\boldsymbol{x}) = u(\boldsymbol{x};\theta_t)$. For any input $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^{d_0}$, the NTK for the output layer of the neural network at the training time t is defined as:

$$K_{\theta_t}(\boldsymbol{x},\boldsymbol{x}') = \nabla_\theta u(\boldsymbol{x};\theta_t)\nabla_\theta u(\boldsymbol{x}';\theta_t)^T \in \mathbb{R}^{d_3\times d_3}. \tag{2}$$

The NTK converges to a deterministic kernel by gradient descent at an infinite width limit and an infinitesimal learning rate during training (Jacot et al., 2018). Given some training data $D = \{X, H\}$ with $X = \{\boldsymbol{x}_i\}_{i=1}^N$ and $H = \{h_i\}_{i=1}^N$, and defined the empirical ℓ^2 loss $\mathcal{L}(\theta) = \frac{1}{2}\|u(X;\theta) - H\|_2^2$, where $u(X;\theta)$ is the network output, H is the desired output. Consider minimizing $\mathcal{L}(\theta)$ by gradient descent with an infinitesimal learning rate, it can generate a continuous form known as gradient flow (Jacot et al., 2018):

$$d\theta_t = -\nabla_\theta \mathcal{L}(\theta_t)dt. \tag{3}$$

The network output $u(\boldsymbol{x};\theta)$ satisfies the following evolution

$$\begin{aligned}\frac{du(X,\theta_t)}{dt} &= -\nabla_\theta u(X;\theta_t)\nabla_\theta u(X;\theta_t)^T \nabla_{u(X;\theta_t)}\mathcal{L}(\theta_t)\\ &= -K_t(X,X)\nabla_{u(X;\theta_t)}\mathcal{L}(\theta_t),\end{aligned} \tag{4}$$

where $K_t(X,X) = \nabla_\theta u(X;\theta_t)\nabla_\theta u(X;\theta_t)^T$ is the NTK matrix at the training time t.

3. Proposed method for forward problem

The PINNs algorithm has garnered attention for its remarkable ability to solve forward and inverse problems related to PDEs. Here, we design a loss-driven dynamic weight algorithm tailored to the characteristics of the PINN algorithm, aimed at solving forward problem involving high-dimensional, time-varying nonlinear PDE with higher-order terms. In this section, we first introduce the form of the high-dimensional Kadomtsev-Petviashvili equation. Then, we review the PINN method. Further, we introduce the proposed method. Finally, we carry out experimental studies on different examples. Meanwhile, for the proposed algorithm, we also conduct a comprehensive hyper-parameter choice analysis.

3.1. Kadomtsev-Petviashvili equation

The Kadomtsev-Petviashvili equation, which is a high-dimensional nonlinear PDE with high-order terms, is used to describe fluid dynamics and plasma physics (Ablowitz & Segur, 1979; Kadomtsev & Petviashvili, 1970; Ma, 2011; Senatorski & Infeld, 1996). It can be written as

$$\frac{\partial^2 u}{\partial x\partial t} - \alpha\left(\frac{\partial u}{\partial x}\right)^2 - \alpha u\frac{\partial^2 u}{\partial x^2} - \beta\frac{\partial^4 u}{\partial x^4} + \gamma\frac{\partial^2 u}{\partial y^2} + \gamma\frac{\partial^2 u}{\partial z^2} = 0, \tag{5}$$

where the solution $u(x,y,z,t)$ can describe many physical wave phenomena at the location (x,y,z) and the time t, and $\alpha,\beta,\gamma \in \mathbb{R}$ are three parameters. Besides, the initial-boundary value conditions of Eq. (5) are defined as follows:

$$\begin{cases} u(x,y,z,t_0) = u_0(x,y,z),\ x\in[x_{lb},x_{ub}],\ y\in[y_{lb},y_{ub}],\ z\in[z_{lb},z_{ub}],\\ u(x_b,y,z,t) = u_b(y,z,t),\ x_b = x_{lb}\ \ and\ \ x_b = x_{ub},\ t\in[t_0,t_b],\\ u(x,y_b,z,t) = u_b(x,z,t),\ y_b = y_{lb}\ \ and\ \ y_b = y_{ub},\ t\in[t_0,t_b],\\ u(x,y,z_b,t) = u_b(x,y,t),\ z_b = z_{lb}\ \ and\ \ z_b = z_{ub},\ t\in[t_0,t_b]. \end{cases} \tag{6}$$

Note that the *forward problem* is to approximate the solution of Eq. (5) in the given parameters (α,β,γ) by the PINN method (see Section 3 for more details). In contrast, the *inverse problem* is to estimate the parameters (α,β,γ) of Eq. (5) using the PINN method (see Section 4 for more details). Please refer to Fig. 1 for the diagram of solving the forward and inverse problems of Eq. (5) by the PINN method. Especially, we mainly focus on the forward problem in this section.

3.2. Physics-informed neural network

This subsection first reviews the PINN method using Eq. (5) as an example, and define the residual of Eq. (5) as

$$f := \frac{\partial^2 u}{\partial x\partial t} - \alpha\left(\frac{\partial u}{\partial x}\right)^2 - \alpha u\frac{\partial^2 u}{\partial x^2} - \beta\frac{\partial^4 u}{\partial x^4} + \gamma\frac{\partial^2 u}{\partial y^2} + \gamma\frac{\partial^2 u}{\partial z^2}.$$

To exploit the neural network model to approximate the solution of Eq. (5), we embed the residual of Eq. (5) in the empirical risk loss with the help of automatic differentiation (Baydin, 2018). Minimizing the total loss generated by the initial-boundary condition and the residual of Eq. (5), that is, to learn the trainable parameters $\theta = \{\omega, b\}$ of the neural network

$$\mathcal{L}(\theta) = \mathcal{L}_0(\theta) + \mathcal{L}_b(\theta) + \mathcal{L}_f(\theta), \tag{7}$$

with

$$\mathcal{L}_0(\theta) = \frac{1}{2}\sum_{n=1}^{N_0}|u(\boldsymbol{x}_0^n;\theta) - u_0(\boldsymbol{x}_0^n)|^2,$$

$$\mathcal{L}_b(\theta) = \frac{1}{2}\sum_{n=1}^{N_b}|u(\boldsymbol{x}_b^n;\theta) - u_b(\boldsymbol{x}_b^n)|^2,$$

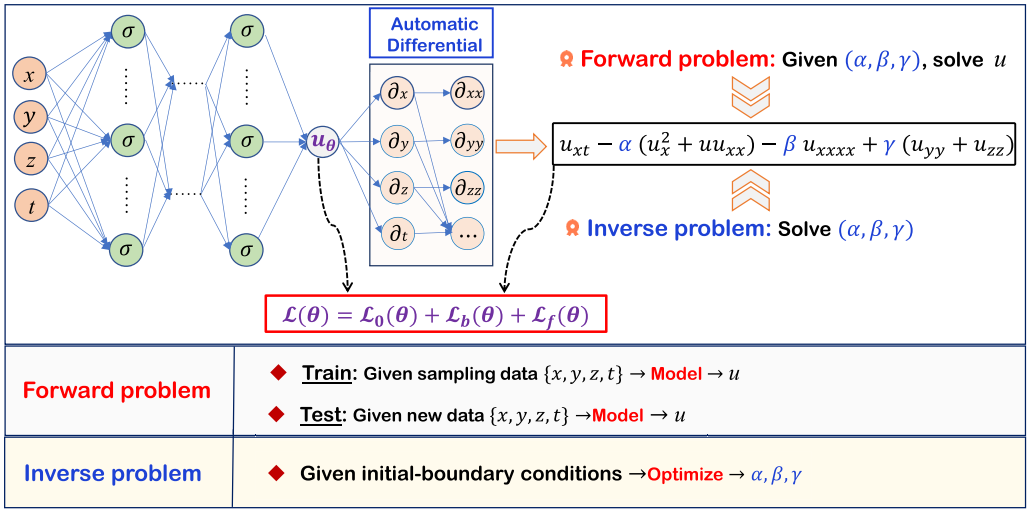


Fig. 1. The diagram of solving the forward and inverse problems of Eq. (5) based on the PINN method.

$$\mathcal{L}_f(\theta) = \frac{1}{2}\sum_{n=1}^{N_f} |f(\boldsymbol{x}_f^n;\theta)|^2.$$

where $\boldsymbol{x} = (x, y, z, t) \in \mathbb{R}^4$, and the loss $\mathcal{L}_0(\theta)$ is generated for the dataset $\{\boldsymbol{x}_0^n, u_0(\boldsymbol{x}_0^n)\}_{n=1}^{N_0}$ that comprises training points sampled from the boundary value domain, and the loss $\mathcal{L}_b(\theta)$ is generated relative to the dataset $\{\boldsymbol{x}_b^n, u_b(\boldsymbol{x}_b^n)\}_{n=1}^{N_b}$ that comprises training points sampled from the boundary value domain. In the meanwhile, the loss $\mathcal{L}_f(\theta)$ is related to the residual of the equation and the dataset $\{\boldsymbol{x}_f^n\}_{n=1}^{N_f}$ consisting of training points sampled from the whole domain. The real numbers of N_0, N_b, and N_f represent the batch size of samples on the initial domain, the boundary domain, and the whole domain, respectively. Minimizing the loss function, using the gradient descent with an infinitesimally small learning rate, can generate a continuous-time gradient flow:

$$\begin{aligned}\frac{d\theta(t)}{dt} &= -\nabla_\theta \mathcal{L}(\theta) \\ &= -\left(\frac{\partial \mathcal{L}_0(\theta)}{\partial\theta} + \frac{\partial \mathcal{L}_b(\theta)}{\partial\theta} + \frac{\partial \mathcal{L}_f(\theta)}{\partial\theta}\right).\end{aligned} \tag{8}$$

Note that here t is not a time but a continuous-time index.

3.3. Physics-informed neural network with dynamic weight

The optimization objective of PINNs comprises multiple loss terms, necessitating a trade-off between conflicting training objectives for each component. This undoubtedly calls for the introduction of an adaptive weighting strategy to mitigate the conflict. Of course, a satisfactory adaptive weight PINN algorithm should have the following characteristics:

- It is capable of stably adjusting the weights assigned to different loss components throughout the training process, avoiding frequent oscillations.
- It does not introduce any supplementary computational cost, ensuring a cost-effective operation.
- There is no indication of memory overflow issues arising during its implementation.

Considering these factors, we propose a loss-driven dynamic weight algorithm based on the optimized properties of PINNs, for solving high-dimensional time-varying PDE with high-order terms, as shown in Algorithm 1.

For the proposed algorithm, namely Algorithm 1, we provide the descriptions of hyperparameters such as a and η.

- $\boldsymbol{a}$ (Weighting factor): The hyperparameter a serves as the weighting factor that balances the contribution of the previous step's dynamic weight and the current step's dynamic weight in the algorithm. In essence, a determines the extent to which the algorithm "remembers" or "forgets" the historical information as it progresses through the training steps. The choice of a is crucial as it directly impacts the stability and convergence of the training process. A value of a close to 0 indicates that the algorithm places a high emphasis on the previous step's weight, potentially leading to the dynamic weight to remain at the initial values. Conversely, a value of a close to 1 makes the algorithm more responsive to the current step's ratio.
- $\boldsymbol{\eta}$ (Learning rate for parameters): The hyperparameter η is the learning rate for the learnable parameters in a neural network model. It controls the step size at each iteration when updating these parameters based on the gradient of the loss function. The learning rate η is a fundamental hyperparameter in many optimization algorithms, and its choice significantly affects the convergence speed and final performance. A large η can lead to overshooting the optimal solution, while a small η may result in slow convergence or getting stuck in local minima. We employ various learning rate decay strategies (e.g., exponential, cosine, piece-wise constant) to dynamically adjust η during training.

Algorithm 1 Physics-Informed Neural Network with Dynamic Weight.

Input: Training data $\{\boldsymbol{x}_0^n, u_0(\boldsymbol{x}_0^n)\}_{n=1}^{N_0}$, $\{\boldsymbol{x}_b^n, u_b(\boldsymbol{x}_b^n)\}_{n=1}^{N_b}$, $\{\boldsymbol{x}_f^n\}_{n=1}^{N_f}$
Output: Trained neural network $u_\theta(\boldsymbol{x})$ approximating the solution to Eq. (5)

1: Redefine the loss function as:

$$\mathcal{L}(\theta) = \mathcal{L}_0(\theta) + \lambda_b \mathcal{L}_b(\theta) + \mathcal{L}_f(\theta),$$

where $\mathcal{L}_0(\theta)$ and $\mathcal{L}_b(\theta)$ are generated from initial-boundary value condition, $\mathcal{L}_f(\theta)$ denotes the residual loss.
2: Initialize $\lambda_b = 1$, $a = 0.1$, $\eta = 10^{-4}$, the maximum iteration "Maxiter".
3: **while** $n \le$ Maxiter **do**
4: **if** $n \,\%\, 10 = 0$ **then**
5: Compute $\hat{\lambda}_b = \frac{\mathcal{L}_b(\theta_n)}{\mathcal{L}_0(\theta_n)}$, where θ_n represents the values of the parameter θ at the n-th iteration
6: Update the weight $\lambda_b \leftarrow (1-a)\lambda_b + a\hat{\lambda}_b$
7: **end if**
8: Update the parameters θ by the gradient descent or it's variant $\theta_{n+1} \leftarrow \theta_n - \eta\nabla_\theta \mathcal{L}_0(\theta_n) - \eta\lambda_b \nabla_\theta \mathcal{L}_b(\theta_n) - \eta\nabla_\theta \mathcal{L}_f(\theta_n)$.
9: **end while**
10: **return** Trained neural network $u_\theta(\boldsymbol{x})$

When solving the time-dependent solutions of Eq. (5) uses the PINN method combined with the adaptive moment estimation algorithm (Adam) (Kingma & Ba, 2014), the loss associated with the initial values decreases the fastest. To alleviate the pathological interferences among various loss components, we adhere to a heuristic principle: dynamic weight should be exclusively applied to the slowest-decaying loss component, while omitting those components exhibiting the steepest descent during training. This is different from the method described in Ref. Yuan et al. (2022). Therefore, the proposed loss function with loss-driven dynamic weights is of the form

$$\mathcal{L}(\theta) = \mathcal{L}_0(\theta) + \lambda_b \mathcal{L}_b(\theta) + \mathcal{L}_f(\theta). \tag{9}$$

Build a neural network with two hidden layers, whose output is $u(\boldsymbol{x};\theta) \in \mathbb{R}^1$. Given the training data, namely $\{\boldsymbol{x}_0^n, u_0(\boldsymbol{x}_0^n)\}_{n=1}^{N_0}$, $\{\boldsymbol{x}_b^n, u_b(\boldsymbol{x}_b^n)\}_{n=1}^{N_b}$, $\{\boldsymbol{x}_f^n\}_{n=1}^{N_f}$, $u(\boldsymbol{x};\theta)$ obeys the following evolution process:

$$\frac{du(\boldsymbol{x}_0;\theta(t))}{dt} = \frac{du(\boldsymbol{x}_0;\theta(t))}{d\theta}^T \frac{d\theta}{dt}, \tag{10}$$

$$\frac{du(\boldsymbol{x}_b;\theta(t))}{dt} = \frac{du(\boldsymbol{x}_b;\theta(t))}{d\theta}^T \frac{d\theta}{dt}, \tag{11}$$

$$\frac{df(\boldsymbol{x}_f;\theta(t))}{dt} = \frac{df(\boldsymbol{x}_f;\theta(t))}{d\theta}^T \frac{d\theta}{dt}. \tag{12}$$

Based on the gradient flow (Eq. 3) and evolutions Eqs. (10)–(12), we can acquire the following lemma.

Lemma 1. *Under the gradient flow (Eq. 8) and the loss (Eq. 9), $u(\boldsymbol{x}_0;\theta(t)), u(\boldsymbol{x}_b;\theta(t))$ and $f(\boldsymbol{x}_f;\theta(t))$ obey the following evolution*

$$\begin{bmatrix} \frac{du(\boldsymbol{x}_0;\theta(t))}{dt} \\ \frac{du(\boldsymbol{x}_b;\theta(t))}{dt} \\ \frac{df(\boldsymbol{x}_f;\theta(t))}{dt} \end{bmatrix} = -\begin{bmatrix} \boldsymbol{K}_0(t) & \lambda_b \boldsymbol{K}_{0b}(t) & \boldsymbol{K}_{0f}(t) \\ \boldsymbol{K}_{b0}(t) & \lambda_b \boldsymbol{K}_b(t) & \boldsymbol{K}_{bf}(t) \\ \boldsymbol{K}_{f0}(t) & \lambda_b \boldsymbol{K}_{fb}(t) & \boldsymbol{K}_f(t) \end{bmatrix} \cdot \begin{bmatrix} u(\boldsymbol{x}_0;\theta(t)) - u_0(\boldsymbol{x}_0) \\ u(\boldsymbol{x}_b;\theta(t)) - u_b(\boldsymbol{x}_b) \\ f(\boldsymbol{x}_f;\theta(t)) \end{bmatrix}$$

where $\boldsymbol{K}_{0b}(t) = \boldsymbol{K}_{b0}^{\mathrm{T}}(t) \in \mathbb{R}^{N_0 \times N_b}$, $\boldsymbol{K}_{0f}(t) = \boldsymbol{K}_{f0}^{\mathrm{T}}(t) \in \mathbb{R}^{N_0 \times N_f}$, $\boldsymbol{K}_{bf}(t) = \boldsymbol{K}_{fb}^{\mathrm{T}}(t) \in \mathbb{R}^{N_b \times N_f}$, $\boldsymbol{K}_0(t) \in \mathbb{R}^{N_0 \times N_0}$, $\boldsymbol{K}_b(t) \in \mathbb{R}^{N_b \times N_b}$, $\boldsymbol{K}_f(t) \in \mathbb{R}^{N_f \times N_f}$,

$$\begin{aligned}(\boldsymbol{K}_0)_{ij}(t) &= \left\langle \frac{du(\boldsymbol{x}_0^i;\theta_t)}{d\theta}, \frac{du(\boldsymbol{x}_0^j;\theta_t)}{d\theta} \right\rangle, (\boldsymbol{K}_b)_{ij}(t) \\ &= \left\langle \frac{du(\boldsymbol{x}_b^i;\theta_t)}{d\theta}, \frac{du(\boldsymbol{x}_b^j;\theta_t)}{d\theta} \right\rangle.\end{aligned}$$

$$\left(\boldsymbol{K}_f\right)_{ij}(t) = \left\langle \frac{df\left(\boldsymbol{x}_f^i;\theta_t\right)}{d\theta}, \frac{df\left(\boldsymbol{x}_f^j;\theta_t\right)}{d\theta} \right\rangle, \left(\boldsymbol{K}_{0b}\right)_{ij}(t)$$
$$= \left\langle \frac{du\left(\boldsymbol{x}_0^i;\theta_t\right)}{d\theta}, \frac{du\left(\boldsymbol{x}_b^j;\theta_t\right)}{d\theta} \right\rangle,$$
$$\left(\boldsymbol{K}_{0f}\right)_{ij}(t) = \left\langle \frac{du\left(\boldsymbol{x}_0^i;\theta_t\right)}{d\theta}, \frac{df\left(\boldsymbol{x}_f^j;\theta_t\right)}{d\theta} \right\rangle, \left(\boldsymbol{K}_{bf}\right)_{ij}(t)$$
$$= \left\langle \frac{du\left(\boldsymbol{x}_b^i;\theta_t\right)}{d\theta}, \frac{df\left(\boldsymbol{x}_f^j;\theta_t\right)}{d\theta} \right\rangle.$$

The proof of Lemma 1 is given in Appendix A.

Remark 1. Following Ref. Wang et al. (2022), based on Lemma 1, the corresponding neural tangent kernel for PINN with loss-driven dynamic weight is as follows:

$$\boldsymbol{K}(t) = \begin{bmatrix} \boldsymbol{K}_0(t) & \lambda_b \boldsymbol{K}_{0b}(t) & \boldsymbol{K}_{0f}(t) \\ \boldsymbol{K}_{b0}(t) & \lambda_b \boldsymbol{K}_b(t) & \boldsymbol{K}_{bf}(t) \\ \boldsymbol{K}_{f0}(t) & \lambda_b \boldsymbol{K}_{fb}(t) & \boldsymbol{K}_f(t) \end{bmatrix}.$$

Observing that the matrix $\boldsymbol{K}(t)$ in Remark 1 is a positive semi-definite matrix and has only one variable parameter λ_b except the network output $u(x, y, z, t)$, namely, assigning a single dynamic weight in the loss (Eq. 9). We speculate that this matrix $\boldsymbol{K}(t)$ is intuitively easier to reach a steady state with respect to other strategies that assign multiple dynamic weights to some extent.

3.4. Experiments

Inspired by comprehensive studies on sampling methods of PINN (Wu et al., 2023) and activation functions (Dashtbayaz et al., 2024; Wang et al., 2023), we propose a dynamic weighting method combined with several optimization algorithms and learning rate decay strategies to be applied to the time-varying solutions of high-dimensional equations.

3.4.1. Case 1

Here, we select a numerical example that exhibits a particular physical phenomenon described by Eq. (5), characterized by its time-varying nature. This phenomenon resembles the evolution of water waves, showcasing dynamic changes over time. The ground truth (GT) solution and coefficients of both high-order and nonlinear terms are given in Ref. Zhang et al. (2021). The spatio-temporal domains are taken as $(x, y, z, t) \in [-5, 5] \times [-5, 5] \times [-0.5, 0.5] \times [0, 1]$. Specifically, we consider first-order optimization algorithms, namely stochastic gradient descent (SGD), adaptive gradient algorithm (Adagrad) (Duchi et al., 2011), RMSprop (Tieleman et al., 2012), Adam, and learning rate annealing (LRA) (Wang et al., 2021). Specifically, LRA is proposed as an adaptive way to tune the weights automatically (i.e., λ_0 and λ_b), that is, the weights of each loss component $[\mathcal{L}_0(\theta)$ and $\mathcal{L}_b(\theta)]$ through using the back-propagated gradient information during model training (Wang et al., 2021):

$$\hat{\lambda}_i = \frac{\max_{\theta_n}\left\{\left|\nabla_\theta \mathcal{L}_f\left(\theta_n\right)\right|\right\}}{\left|\nabla_\theta \lambda_i \mathcal{L}_i\left(\theta_n\right)\right|}, \ \lambda_i = (1-\alpha)\lambda_i + \alpha\hat{\lambda}_i \ (i = 0, b), \ \lambda_f = 1.$$

At the same time, we also consider the constant learning rate and several learning rate decay strategies, which decay according to the number of iterations, such as piecewise constant decay, exponential decay, and cosine decay. The corresponding hyperparameter selections are shown in Table 1, and it should be noted that the settings are the same for each experiment in this subsection.

The relative L_2 errors and running time of the results obtained by the PINN method under different optimization algorithms and learning rates are shown in Table 2 under the maximum number of iterations.

Table 1
Hyperparameter selection for case 1.

Hidden layer	2×100
Activation function	Hyperbolic Tangent
Train data	$N_0 = 3k, N_b = 12k, N_f = 20k$
Iterations	20001
Initialization	Xavier (Glorot & Bengio, 2010)
Collocation points sample	Latin Hypercube Sampling (Stein, 1987)

Table 2
The relative L_2 error and running time for case 1 by different optimization algorithms and learning rate strategies. (Time: Seconds).

		constant	piece-wise	exponential	cosine
PINN(SGD)	Rel. L_2	9.0654e-01	8.9840e-01	9.0457e-01	8.9618e-01
	Time	580.932	576.589	576.348	576.761
PINN(Adagrad)	Rel. L_2	9.0091e-01	8.1487e-01	8.4938e-01	8.1637e-01
	Time	580.608	581.032	581.332	580.925
PINN(RMSprop)	Rel. L_2	2.4576e-01	9.5632e-02	1.5938e-01	1.3342e-01
	Time	580.793	580.138	580.212	580.886
PINN(Adam)	Rel. L_2	6.6042e-02	5.6252e-02	8.0045e-02	7.0447e-02
	Time	585.165	584.654	585.191	584.997
PINN(LRA)	Rel. L_2	4.3868e-02	2.5712e-02	**4.6435e-02**	3.3154e-02
	Time	630.655	632.927	632.966	632.027
Ours	Rel. L_2	**2.7287e-02**	**1.9080e-02**	5.6457e-02	**2.0381e-02**
	Time	588.296	587.235	587.230	587.900

Among them, the update frequency of the dynamic weight is once every 10 gradient descent steps for both the LRA algorithm and the proposed algorithm.

It can be seen from Table 2 that these learning rate strategies have different degrees of influence on the accuracy of the same optimization algorithm, and their differences in time cost can be ignored so that it can be considered that no additional time cost is generated. We find that the SGD and Adagrad algorithms don't work. The other algorithms combined with the piece-wise constant learning rate strategy show the best results. In particular, our proposed algorithm improves accuracy and avoids adding additional time consumption. This is due not only to dynamic weight mitigating learning challenges at the boundaries, but also to their inherent property of avoiding computational overhead. On the contrary, LRA algorithm improves the accuracy to varying degrees but sacrifices the time cost, which is caused by the calculation of the gradient.

In Fig. 2, we show the update process of the dynamic weights and total losses for the LRA algorithm and our proposed algorithm, which is recorded every 10 gradient descent steps during training. The dynamic weights of the two algorithms are shown in the first row of Fig. 2, where the red line shows that our algorithm can be smooth and tends to a stable state after a certain number of steps. Whereas the dynamic weights (the blue line and the orange line) of the LRA algorithm are oscillating and unstable. This phenomenon is also reflected in the loss curve at the bottom of Fig. 2.

Meanwhile, we present the visualized result in Fig. 3, which is the result of combining our proposed algorithm with the piece-wise constant learning rate. We show that our algorithm not only reproduces the solution of high-dimensional PDE with high-order terms but also captures the time-varying characteristics. The top of Fig. 3 shows the density plot of the predicted solution at different moments. We find that the position of the wave peak is moving with time, which is similar to the undulating phase of water waves. To visualize the accuracy of our algorithm, we plot the GT and predicted values at the position of the white dashed line at the top of Fig. 3. It is important to note that our algorithm captures the changing characteristics of waves.

3.4.2. Case 2

In this subsection, we aim to validate the developed theory and examine the effectiveness of the proposed PINN with a dynamic weight

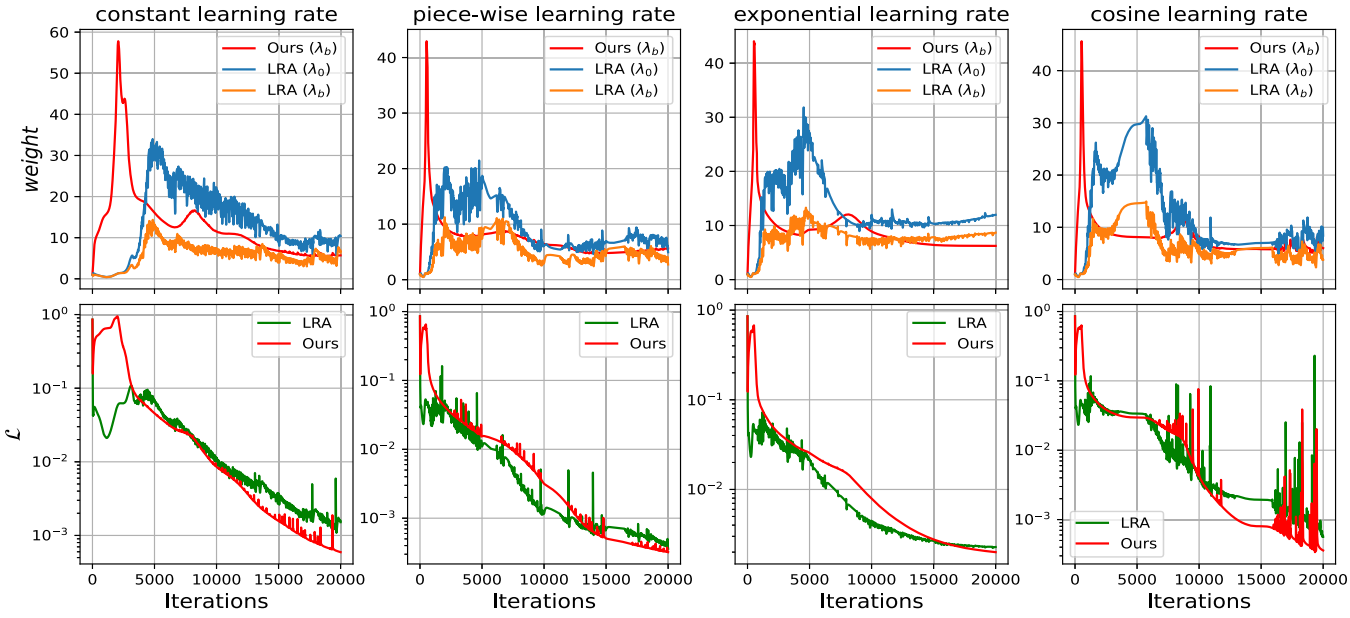


Fig. 2. Losses and dynamic weights recorded during model training through the LRA algorithm and our proposed algorithm for case 1. Top: The dynamic weights value recorded during training. Bottom: Total loss value logged during training.

algorithm on the complex structure whose evolving shape is approximately periodic and whose amplitude changes with time for Eq. (5). Its exact solution and coefficients were reported in Ref. Qian et al. (2016). The spatio-temporal domains are taken as $(x, y, z, t) \in [-5, 5] \times [-5, 5] \times [-0.5, 0.5] \times [-0.6, 0.6]$. Specifically, we still consider first-order optimization algorithms, namely SGD, Adagrad, RMSprop, Adam, and LRA. At the same time, we also consider the constant learning rate and several learning rate decay strategies, which decay according to the number of iterations, such as piecewise constant, exponential, and cosine functions. The corresponding hyperparameter selections are shown

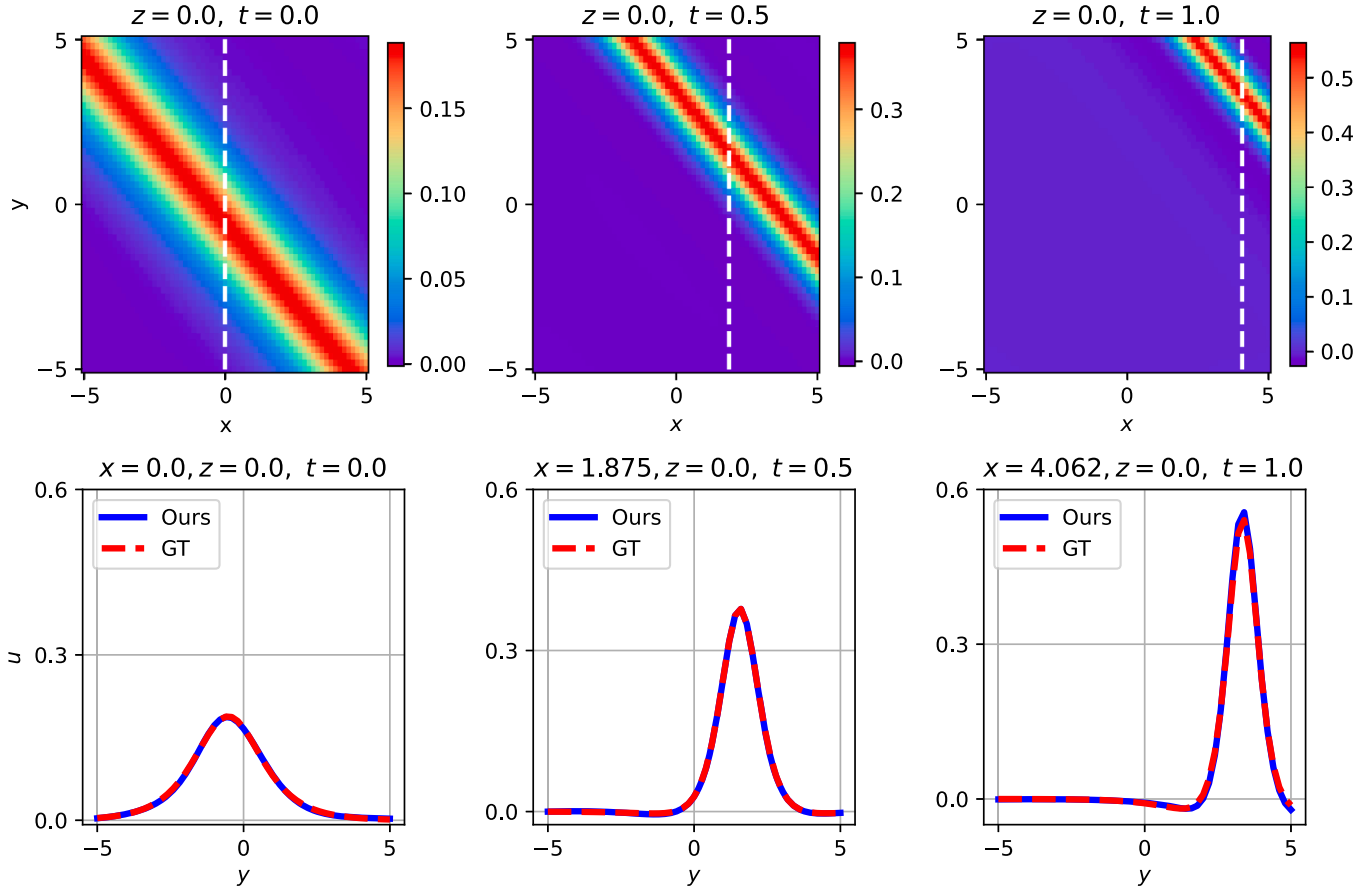


Fig. 3. The prediction result is our algorithm combined with the piecewise constant learning rate for case 1. Top: Density plot with fixed spatial variable z and temporal variable t. Bottom: The evolution of the position of the white dashed line at the top.

Table 3
Hyperparameter selection for case 2.

Hidden layer	5×40
Activation function	Hyperbolic Tangent
Train data	$N_0 = 3k, N_b = 12k, N_f = 20k$
Iterations	20001
Initialization	Xavier (Glorot & Bengio, 2010)
Collocation points sample	Latin Hypercube Sampling (Stein, 1987)

Table 4
The relative L_2 errors and running time for case 2 by different optimization algorithms and learning rates strategies. (Time: Seconds).

		constant	piece-wise	exponential	cosine
PINN(SGD)	Rel. L_2	1.0003e+00	1.0010e+00	1.0001e+00	1.0038e+00
	Time	810.814	808.042	808.616	807.488
PINN(Adagrad)	Rel. L_2	9.9991e-01	1.0481e+00	1.0413e+00	1.0493e+00
	Time	817.636	819.183	819.322	818.650
PINN(RMSprop)	Rel. L_2	9.4944e-01	8.2996e-01	9.0864e-01	8.4174e-01
	Time	818.525	816.101	816.664	818.637
PINN(Adam)	Rel. L_2	7.0389e-01	6.3449e-01	7.3711e-01	6.8698e-01
	Time	825.806	827.048	826.865	828.498
PINN(LRA)	Rel. L_2	2.1366e-01	3.6056e-01	4.7539e-01	3.8374e-01
	Time	883.594	882.568	886.074	883.914
Ours	Rel. L_2	**3.8572e-02**	**1.7020e-01**	**2.0425e-01**	**1.4566e-01**
	Time	825.813	826.814	820.336	822.940

in Table 3, and it should be noted that the settings are the same for each experiment in this subsection.

To measure the performance of these algorithms, the relative L_2 errors and running time of the results obtained by the PINN method under different optimization algorithms and learning rate strategies are shown in Table 4 under the maximum number of iterations. Again, these algorithms with dynamic weights are updated every 10 gradient descent steps.

As can be seen from Table 4 we obtain unsatisfactory results by using SGD, Adagrad, RMSprop, and Adam optimization algorithms, which indicates that these optimization algorithms are insufficient to represent the characteristics of this structure. By adopting a constant learning rate, our proposed algorithm is not only superior to the LRA algorithm in terms of accuracy but also has advantages in terms of time consumption. We speculate that these depend on the dynamic weights being assigned key parts and only involving multiplication operations, not calculating gradients. To be precise, our algorithm has advantages in both precision and time for the relatively simple time-varying case (Case 1) and complex physical phenomena (Case 2).

In addition, to evaluate the stability of these algorithms in the optimization process, we record the dynamic weights and loss changes of the two algorithms, in Fig. 4. In the top, we find that the dynamic weights of the LRA algorithm fluctuate sharply, and this phenomenon is also reflected in the loss curve at the bottom of Fig. 4. We observe that the dynamic weight of our proposed algorithm tends to stabilize after a certain number of steps, and the loss curve is smoother than that of LRA.

We present the absolute error of both LRA and the proposed algorithms at several time in Fig. 5. In (a1)–(a3), we show the density plot of GT value at different time, where the dark lines evolve from wide to narrow to wide, which is the change in amplitude from low to high to low. Intuitively, this phenomenon could be understood as the appearance of a plane to a water wave in the ocean, and finally, the attenuation of energy makes the water wave disappear. In (b1)–(b3), we provide the absolute error for the LRA algorithm with the constant learning rate, where the dark position indicates that the error is large and occurs at the high amplitude position. We show the absolute error of our algorithm with the constant learning rate at (c1)–(c3). It should be noted that these absolute errors share the same color bar at the same time, which could be seen in our algorithm showing advantages in the prediction of complex physical phenomena.

3.5. Choices analysis for hyperparameters

Although our algorithm demonstrates higher accuracy and equal time cost than other algorithms in Cases 1 and 2, there are still several potential issues worth further exploring, such as the update frequency and allocation principle of dynamic weight. Specifically, we introduce two hyperparameters based on Algorithm 1, namely the dynamic weight allocation principle S and the dynamic weight update frequency P, which may provide a more comprehensive analysis and explanation for the proposed algorithm. The allocation principle S refers to the dynamic

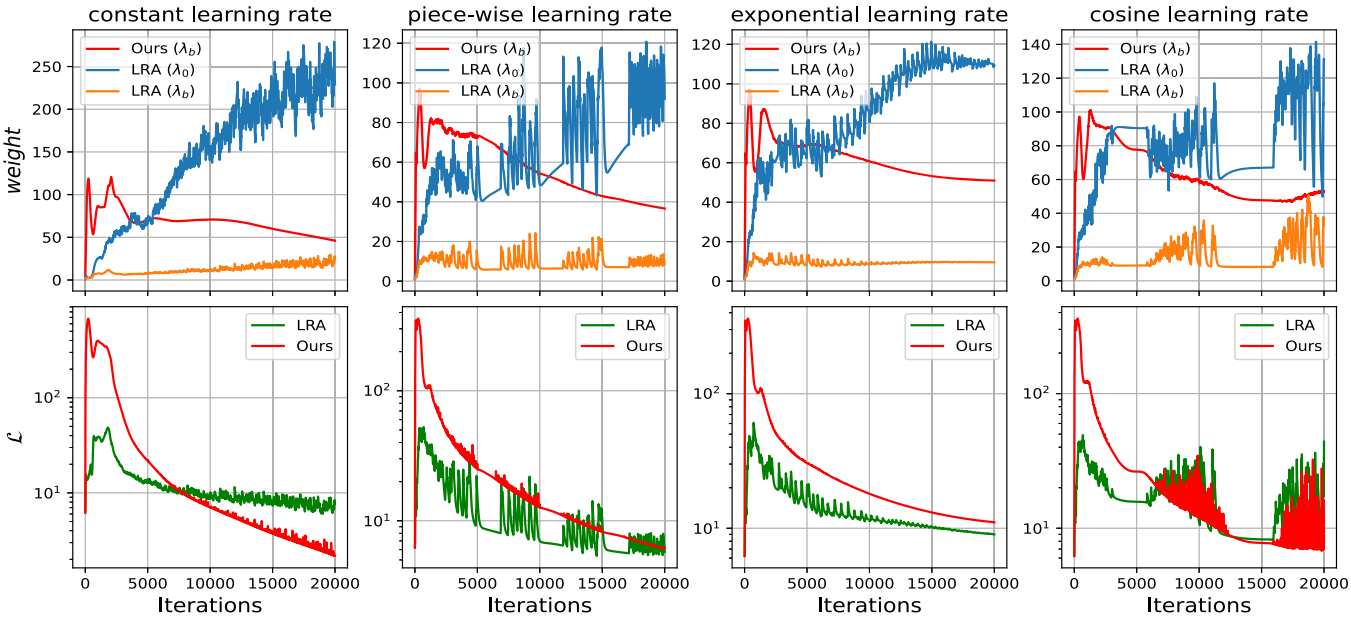


Fig. 4. Losses and dynamic weights recorded during model training through the LRA algorithm and our proposed algorithm for case 2. Top: The dynamic weights value recorded during training. Bottom: Total loss value logged during training. Loss values and dynamic weights are recorded every 10 steps during training.

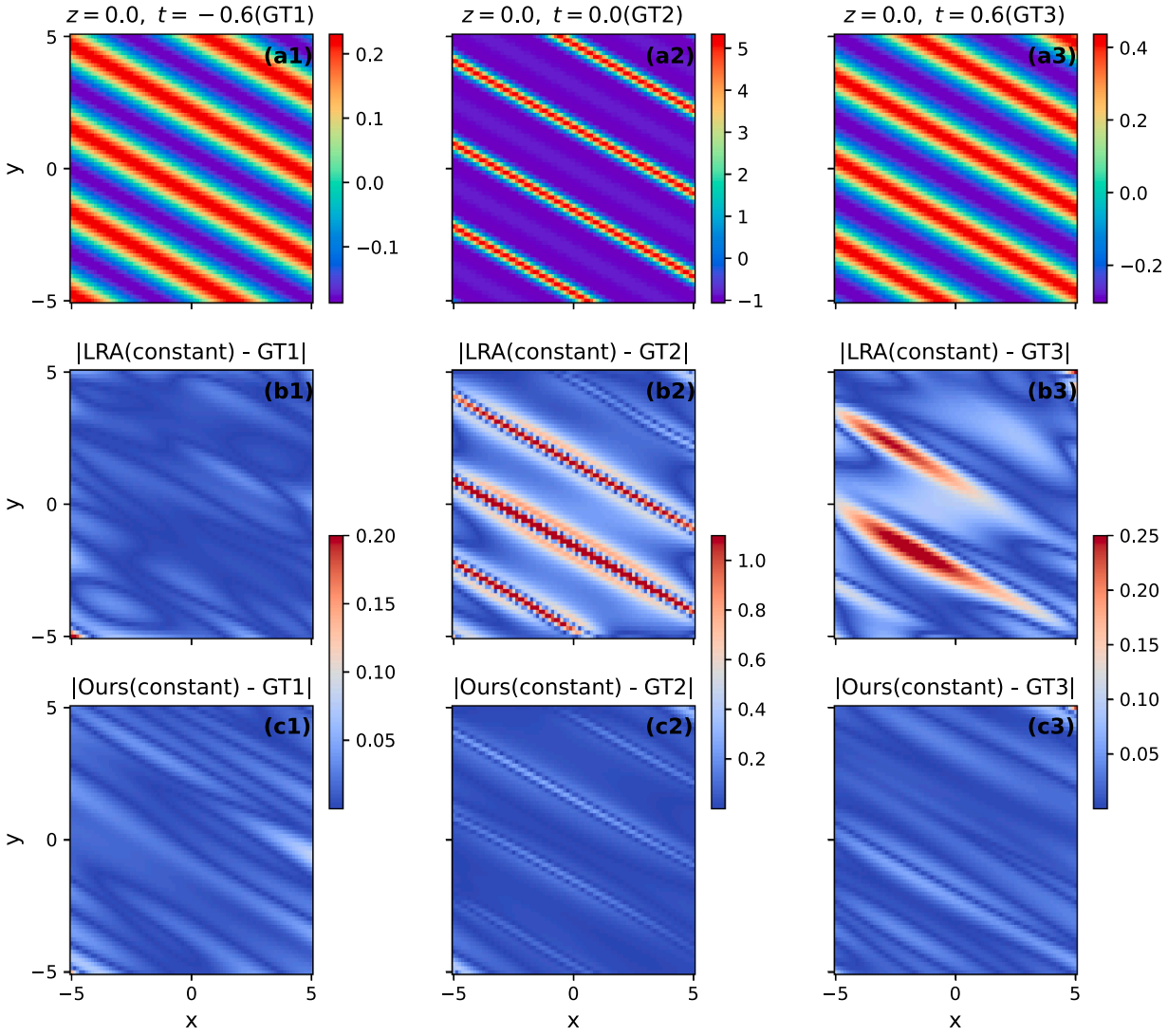


Fig. 5. Absolute error of both the LRA algorithm and our proposed algorithm at different time points for case 2. (a1)-(a3): Density plot of GT values with fixed spatial variable z and temporal variable t. (b1)-(b3): Absolute error of the LRA algorithm with a constant learning rate. (c1)-(c3): Absolute error of our proposed algorithm with a constant learning rate.

weight λ_b being updated starting from the gradient descent step S during the model training process. Similarly, the update frequency P refers to the dynamic weight λ_b being updated every P gradient descent step during model training. For a clearer understanding, we provide a modified version based on the allocation principle and update frequency in Algorithm 2. **Allocation principle S.** In Table 5, we report the relative L_2 error and running time based on case 1 under different dynamic weight assignment principles S and learning rate strategies.

From Table 5, we observe that varying assignment principles influence the accuracy of the predicted solution without sacrificing the time cost. This also indicates that the dynamic weight algorithm we proposed can avoid additional time consumption. For this example, we choose constant and piecewise constant learning rates and assign dynamic weights principle $S = 1$ optimal accuracy. If the exponential and the cosine functions are selected as the learning rate, the dynamic weight is set to assign from $S = 3000$ as the optimal combination. The assignment principle $S = 3000$ with the learning rate as a cosine function has the best effect. Here, in addition to the learning rate affecting the accuracy of the algorithm, the assignment principle of dynamic weight also affects the accuracy to varying degrees.

We present that dynamic weight (λ_b) and total loss ($\mathcal{L}$) are updated during training (manually set to update every 10 steps) in different dynamic weight assignment principles in Fig. 6. Among them, we find that the maximum value of dynamic weight exceeds 70 under a constant learning rate when the assignment principle is chosen as $S = 1000$ and $S = 2000$. In addition to the constant learning rate, the maximum value of the dynamic weight decreases with the increase of S. Meanwhile,

Table 5
The relative L_2 error and running time for case 1 by different dynamic weight assignment principles (S) and learning rate strategies using Algorithm 2. (Time: Seconds, update frequency $P = 10$).

		constant	piece-wise	exponential	cosine
$S = 1$	Rel. L_2	**2.7287e-02**	**1.9080e-02**	5.6457e-02	2.0381e-02
	Time	588.296	587.235	587.230	587.900
$S = 1000$	Rel. L_2	3.5347e-02	2.8320e-02	6.6826e-02	3.4552e-02
	Time	587.438	585.225	587.233	586.780
$S = 2000$	Rel. L_2	3.3992e-02	1.9976e-02	4.3619e-02	2.4939e-02
	Time	589.397	585.607	584.967	585.083
$S = 3000$	Rel. L_2	4.1574e-02	2.0155e-02	**4.0955e-02**	**1.8945e-02**
	Time	585.043	585.342	587.834	587.396
$S = 4000$	Rel. L_2	4.2107e-02	2.4467e-02	4.9904e-02	2.1867e-02
	Time	588.079	598.326	594.384	588.003
$S = 5000$	Rel. L_2	3.3950e-02	2.2814e-02	6.6820e-02	2.0410e-02
	Time	587.145	587.432	587.761	587.323

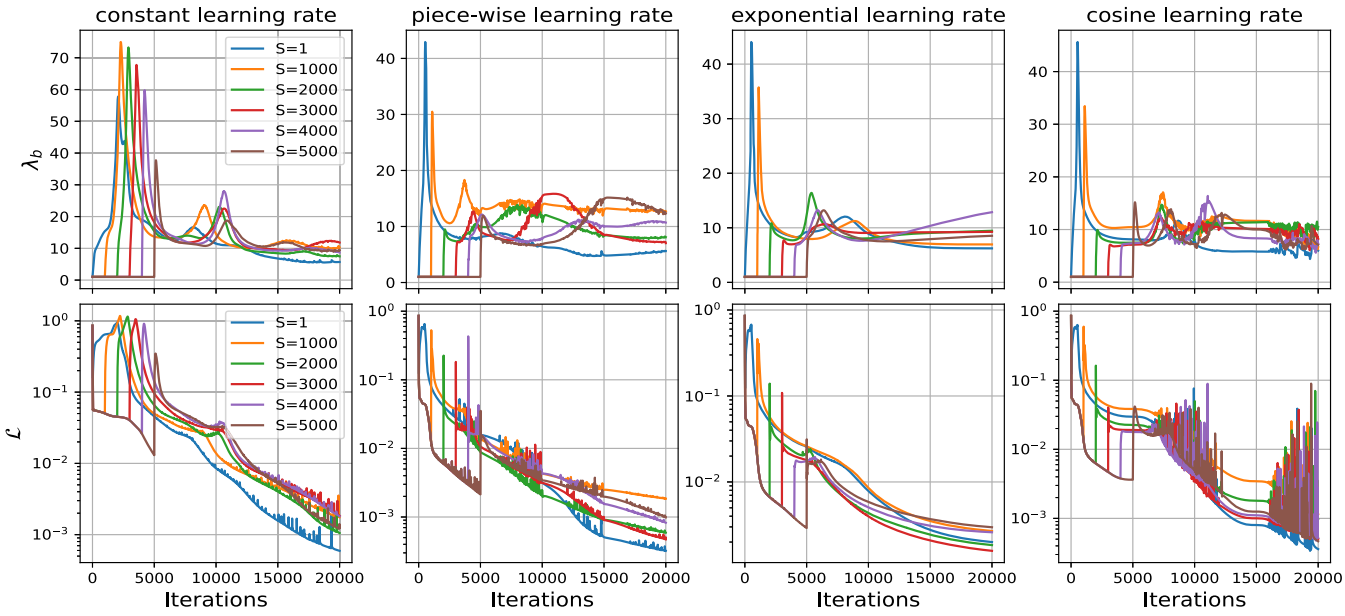


Fig. 6. The change of the loss value $\mathcal{L}$ and the dynamic weight λ_b with the number of iterations in different assignment principles during training. Top: Variation of λ_b under various learning rate strategies based on different dynamic weight assignment principles. Bottom: The change of the loss value $\mathcal{L}$ based on different dynamic weight assignment principles under various learning rate strategies.

Algorithm 2 Modified physics-informed neural network with dynamic weight.

Input: Training data $\{\boldsymbol{x}_0^n, u_0(\boldsymbol{x}_0^n)\}_{n=1}^{N_0}$, $\{\boldsymbol{x}_b^n, u_b(\boldsymbol{x}_b^n)\}_{n=1}^{N_b}$, $\{\boldsymbol{x}_f^n\}_{n=1}^{N_f}$

Output: Trained neural network $u_\theta(\boldsymbol{x})$ approximating the solution to Eq. (5)

1: Redefine the loss function as:

$$\mathcal{L}(\theta) = \mathcal{L}_0(\theta) + \lambda_b \mathcal{L}_b(\theta) + \mathcal{L}_f(\theta),$$

where $\mathcal{L}_0(\theta)$ and $\mathcal{L}_b(\theta)$ are generated from initial-boundary value condition, $\mathcal{L}_f(\theta)$ denotes the residual loss.

2: Initialize $\lambda_b = 1$, $a = 0.1$, $\eta = 10^{-4}$, the allocation principle S, the update frequency P, the maximum iteration "Maxiter".

3: **while** $n \le$ Maxiter **do**

4: **if** $n \,\%\, P = 0$ and $n \ge S$ **then**

5: Compute $\hat{\lambda}_b = \frac{\mathcal{L}_b(\theta_n)}{\mathcal{L}_0(\theta_n)}$, where θ_n represents the values of the parameter θ at the n-th iteration

6: Update the weight $\lambda_b \leftarrow (1-a)\lambda_b + a\hat{\lambda}_b$

7: **end if**

8: Update the parameters θ by the gradient descent or it's variant $\theta_{n+1} \leftarrow \theta_n - \eta\nabla_\theta\mathcal{L}_0(\theta_n) - \eta\lambda_b\nabla_\theta\mathcal{L}_b(\theta_n) - \eta\nabla_\theta\mathcal{L}_f(\theta_n)$.

9: **end while**

10: **return** Trained neural network $u_\theta(\boldsymbol{x})$

the peak value of the dynamic weight with $S = 1$ is the largest, and the peak value of the dynamic weight with different assignment principles also moves backward with the increase of S. Interestingly, the dynamic weight of different assignment principles reaches a peak after a certain number of steps, decays to a certain range, and finally fluctuates within a certain range. In addition, from the change of loss value, it can be found that, under the constant learning rate, the peak value still moves backward with the increase of S. Similarly, there are similar characteristics under other learning rate strategies. Under the constant learning rate, the curve of loss decline has the largest slope intuitively and could continue to decline, while other learning rates are relatively flat but also have a downward trend.

Update frequency P. Similarly, we thoroughly analyze the effect of update frequency (manually set) on the performance of Algorithm 2.

Table 6
The relative L_2 error and running time based on various dynamic weight update frequencies (P) and learning rate strategies for case 1. (Time: Seconds, allocation principles $S = 1$).

		constant	piece-wise	exponential	cosine
$P = 1$	Rel. L_2	3.2234e-02	3.0445e-02	6.8579e-02	2.2571e-02
	Time	603.568	596.824	597.282	596.472
$P = 10$	Rel. L_2	**2.7287e-02**	**1.9080e-02**	**5.6457e-02**	**2.0381e-02**
	Time	588.296	587.235	587.230	587.900
$P = 20$	Rel. L_2	3.6491e-02	3.0830e-02	7.2395e-02	2.4395e-02
	Time	584.923	584.604	585.148	584.557
$P = 30$	Rel. L_2	3.6118e-02	2.7720e-02	7.2222e-02	3.1371e-02
	Time	584.021	584.107	584.395	584.546
$P = 40$	Rel. L_2	3.5271e-02	2.9878e-02	6.6394e-02	2.9237e-02
	Time	585.203	584.511	587.860	588.739
$P = 50$	Rel. L_2	3.5251e-02	3.4928e-02	7.1605e-02	3.5456e-02
	Time	585.782	584.708	589.408	587.436

Table 6 provides insights into how varying the dynamic weight update frequency affects both the accuracy of the solution (measured by the relative L_2 error) and the computational efficiency (indicated by the running time).

Based on the information provided in Table 6, it is evident that varying dynamic weight update frequencies influence the accuracy and efficiency of the algorithm. Among them, the multi-step update is better than the step-by-step update for the dynamic weight, which is one of the very interesting phenomena. In addition, we find that the accuracy fluctuates in a small range by changing the update frequency under the same learning rate. The learning rate plays a major role in the accuracy compared to the update frequency.

Fig. 7 shows the dynamic weight and total losses during model training at different update frequencies and learning rate strategies. We observe that the dynamic weight experiences a fluctuation at the beginning and then is either flat or still has a very small range of oscillations. Among them, fluctuations of dynamic weight almost appear in the same position for different update frequencies. At the same time, the loss changes of different update frequencies do not appear to have

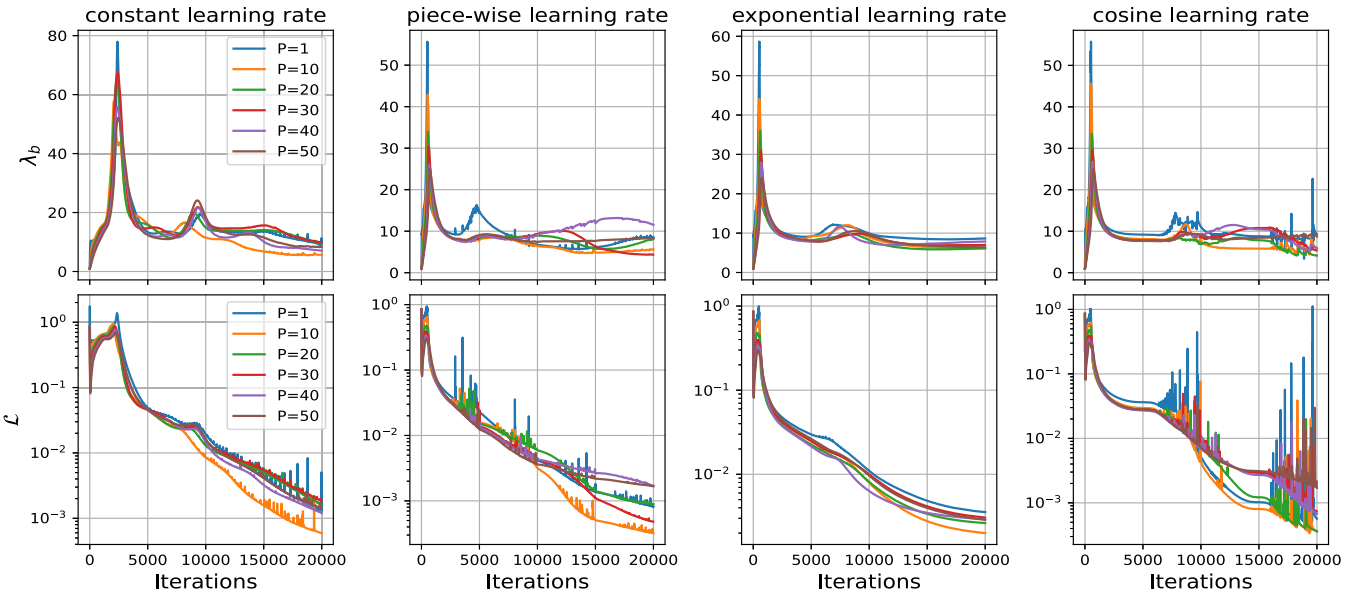


Fig. 7. Changes in loss values $\mathcal{L}$ and dynamic weights λ_b with the number of iterations during training at different update frequencies. Top: Variation of λ_b under various learning rate strategies based on different dynamic weight update frequencies. Bottom: The change of the loss value $\mathcal{L}$ based on different dynamic weight update frequencies under various learning rate strategies.

an obvious dislocation phenomenon, but there are different attenuation speeds. On the contrary, the different assignment principles appear to be an obvious dislocation phenomenon for loss variation but finally tend to converge at the same rate. To reveal in more detail how to choose the update frequency and the assignment principle, we show the dependency between the assignment principle and the update frequency under different learning rate strategies in Appendix B, i.e., Tables B.9–B.12, respectively.

4. Proposed method for inverse problem

Currently, the PINN method also focuses on solving the inverse problem of PDEs. Utilizing PINN to study the inverse problem requires not only the inference of parameters but also the approximation of solutions. There are still convergence issues that merit further systematic investigation into underlying mechanisms. In this section, we concentrate on the inverse problem related to the dynamic analysis of parameters inference in high-dimensional nonlinear equations with high-order terms, which involve multiple parameters and large-scale problems. Naturally, parameter discovery and predicted solutions share the same loss function, so the question arises: how to constrain them? Here, we will examine these issues. In the following content, we first conduct a NTK analysis on the inverse problems of PINN. Subsequently, we develop a novel algorithm, called ResTranPINN. Finally, we carry out multiple sets of experiments.

4.1. PINN and NTK

First, we define the learning criterion that solves the inverse problem using PINN for Eq. (5). This criterion is based on a composite loss function that is constructed from both the initial-boundary value conditions and the residual of Eq. (5),

$$\mathcal{L}(\theta) = \mathcal{L}_0(\theta_1) + \mathcal{L}_b(\theta_1) + \mathcal{L}_f(\theta). \tag{13}$$

The trainable parameters of neural network are denoted by $\theta_1 = \{\boldsymbol{w}, \boldsymbol{b}\}$ and the inference parameters of Eq. (5) are represented as $\theta_2 = \{\alpha, \beta, \gamma\}$. The overall learnable parameters are expressed as $\theta = \theta_1 \cup \theta_2$.

Based on the gradient flow system (Eq. 8), we acquire the following lemma describing the evolution for solving the inverse problem of Eq. (5).

Lemma 2. *Under the gradient flow and loss (Eq. 13), the initial $u(\boldsymbol{x}_0;\theta_1(t))$, the boundary $u(\boldsymbol{x}_b;\theta_1(t))$, the residual $f(\boldsymbol{x}_f;\theta(t))$, parameters α, β and γ obey the following evolution,*

$$\begin{bmatrix}\boldsymbol{U}(t)\\ \boldsymbol{P}(t)\end{bmatrix} = -\begin{bmatrix}\boldsymbol{M}(t) & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{E}\end{bmatrix}\cdot\begin{bmatrix}\boldsymbol{A}(t)\\ \boldsymbol{B}(t)\end{bmatrix},$$

where $\boldsymbol{E}$ is the identity matrix,

$$\boldsymbol{U}(t) = \begin{bmatrix}\frac{du(\boldsymbol{x}_0;\theta_1(t))}{dt}\\ \frac{du(\boldsymbol{x}_b;\theta_1(t))}{dt}\\ \frac{df(\boldsymbol{x}_f;\theta(t))}{dt}\end{bmatrix}, \boldsymbol{P}(t) = \begin{bmatrix}\frac{d\alpha(t)}{dt}\\ \frac{d\beta(t)}{dt}\\ \frac{d\gamma(t)}{dt}\end{bmatrix},$$

$$\boldsymbol{M}(t) = \begin{bmatrix}\boldsymbol{K}_0(t) & \boldsymbol{K}_{0b}(t) & \boldsymbol{K}_{0f}(t)\\ \boldsymbol{K}_{b0}(t) & \boldsymbol{K}_b(t) & \boldsymbol{K}_{bf}(t)\\ \boldsymbol{K}_{f0}(t) & \boldsymbol{K}_{fb}(t) & \boldsymbol{K}_f(t)\end{bmatrix}, \boldsymbol{A}(t) = \begin{bmatrix}u(\boldsymbol{x}_0;\theta_1(t)) - u_0(\boldsymbol{x}_0)\\ u(\boldsymbol{x}_b;\theta_1(t)) - u_b(\boldsymbol{x}_b)\\ f(\boldsymbol{x}_f;\theta(t))\end{bmatrix},$$

$$\boldsymbol{B}(t) = \begin{bmatrix}-f(\boldsymbol{x}_f;\theta(t))^{\mathrm{T}}\frac{\partial}{\partial x}\left(u(\boldsymbol{x}_f;\theta_1(t))\frac{\partial u(\boldsymbol{x}_f;\theta_1(t))}{\partial x}\right)\\ -f(\boldsymbol{x}_f;\theta(t))^{\mathrm{T}}\frac{\partial^4 u(\boldsymbol{x}_f;\theta_1(t))}{\partial x^4}\\ f(\boldsymbol{x}_f;\theta(t))^{\mathrm{T}}\left(\frac{\partial^2 u(\boldsymbol{x}_f;\theta_1(t))}{\partial y^2} + \frac{\partial^2 u(\boldsymbol{x}_f;\theta_1(t))}{\partial z^2}\right)\end{bmatrix}.$$

The proof of Lemma 2 refers to Lemma 1.

Remark 2. According to Lemma 2, we call the matrix $\boldsymbol{K}(t)$ as the neural tangent kernel of the inverse problem for the PINN, which is expressed as:

$$\boldsymbol{K}(t) = \begin{bmatrix}\boldsymbol{M}(t) & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{E}\end{bmatrix},$$

where $\boldsymbol{M}(t)$ is defined in Lemma 2.

We find that the matrix $\boldsymbol{K}(t)$ in Remark 2 is a block diagonal matrix, which is also represented by $\boldsymbol{K}(t) = diag(\boldsymbol{M}(t), \boldsymbol{E})$. This block-diagonal matrix is square, and all the subblocks on the non-principal diagonal are zero matrices. Block diagonal matrices have some useful properties, such as the determinant being the product of the determinants of the principal diagonal matrices. At the same time, it is easy to see that the matrix

$\boldsymbol{K}(t)$ is a semi-positive definite matrix and can therefore be decomposed orthogonally. From the properties of a given matrix, we imply the eigenvalue of the matrix $\boldsymbol{K}(t)$ dominated by $\boldsymbol{M}(t)$ from Remark Eq. 2 under the gradient descent by using PINN to solve the parameters discovery of Eq. (5). The evolution process in which the parameters of Eq. (5) are inferred is shown as follows,

$$\frac{d\alpha(t)}{dt} = f\left(\boldsymbol{x}_f;\theta(t)\right)^{\mathrm{T}}\frac{\partial}{\partial x}\left(u\left(\boldsymbol{x}_f;\theta_1(t)\right)\frac{\partial u\left(\boldsymbol{x}_f;\theta_1(t)\right)}{\partial x}\right),$$

$$\frac{d\beta(t)}{dt} = f\left(\boldsymbol{x}_f;\theta(t)\right)^{\mathrm{T}}\frac{\partial^4 u\left(\boldsymbol{x}_f;\theta_1(t)\right)}{\partial x^4},$$

$$\frac{d\gamma(t)}{dt} = -f\left(\boldsymbol{x}_f;\theta(t)\right)^{\mathrm{T}}\left(\frac{\partial^2 u\left(\boldsymbol{x}_f;\theta_1(t)\right)}{\partial y^2}+\frac{\partial^2 u\left(\boldsymbol{x}_f;\theta_1(t)\right)}{\partial z^2}\right).$$

Based on this, we find that the approximate solution $u(x, y, z, t)$ and the residual of Eq. (5) play a key role in inferring the parameters based on gradient descent, which may also be understood to be dominated by the approximate solution. To infer the multiple parameters, we should develop a robust PINN-based algorithm that achieves high accuracy and efficiency.

4.2. Residual transformation physics-informed neural network

The above theory reveals that the PINN method to identify the parameters of PDEs is dependent on the choice of the optimization algorithm. Multi-parameters discovery of PDEs, where these parameters are between large scales, remains a challenge based on the PINN method. In order to design a more efficient algorithm, we transform the residual with the help of limiting equivalence ($\lim_{x\to 0}\frac{\sin(x)}{x}=1$). In this way, we construct a novel loss function as follows

$$\mathcal{L}(\theta) = \mathcal{L}_0(\theta_1)+\mathcal{L}_b(\theta_1)+\sin(\mathcal{L}_f(\theta)). \tag{14}$$

The generated method induced by the loss function (Eq. 14) is referred to as Residual Transformation Physics-Informed Neural Network (**ResTranPINN**). Under the new learning criterion namely the loss function (Eq. 14), its first-order and second-order gradients are modified as follows, respectively

$$\frac{\partial\mathcal{L}(\theta)}{\partial\theta_1} = \frac{\partial\mathcal{L}_0(\theta_1)}{\partial\theta_1}+\frac{\partial\mathcal{L}_b(\theta_1)}{\partial\theta_1}+\cos(\mathcal{L}_f)\frac{\partial\mathcal{L}_f(\theta)}{\partial\theta_1},$$

$$\frac{\partial^2\mathcal{L}(\theta)}{\partial\theta_1^2} = \frac{\partial^2\mathcal{L}_0(\theta_1)}{\partial\theta_1^2}+\frac{\partial^2\mathcal{L}_b(\theta_1)}{\partial\theta_1^2}+\cos(\mathcal{L}_f)\frac{\partial^2\mathcal{L}_f(\theta)}{\partial\theta_1^2}-\sin(\mathcal{L}_f)\left(\frac{\partial\mathcal{L}_f(\theta)}{\partial\theta_1}\right)^2.$$

Accordingly, the update step size of the gradient descent method is also controlled by $\cos(\mathcal{L}_f)$, which seems to act as an adaptive learning rate. In particular, the second-order gradient of loss function (Eq. 14) is different from the previous form. It is similar to a weighting for the second-order gradient and the square of the first-order gradient, which could alleviate the fluctuation phenomenon. The trainable parameters θ_1 and θ_2 are updated by the gradient descent in the following way

$$\theta_1^{(n+1)} = \theta_1^{(n)}-\eta\frac{\partial\mathcal{L}_0(\theta_1)}{\partial\theta_1}-\eta\frac{\partial\mathcal{L}_b(\theta_1)}{\partial\theta_1}-\eta\cos(\mathcal{L}_f)\frac{\partial\mathcal{L}_f(\theta)}{\partial\theta_1},$$

$$\theta_2^{(n+1)} = \theta_2^{(n)}-\eta\cos(\mathcal{L}_f)\frac{\partial\mathcal{L}_f(\theta)}{\partial\theta_2}.$$

We observe that a term $\cos(\mathcal{L}_f)$, which in the domain $[0, 1]$ is a decreasing function, that controls how much the trainable parameter changes. As the residual loss $\mathcal{L}_f$ value nears $\pi/2$, the cosine function $\cos(\mathcal{L}_f)$ diminishes to an infinitesimally small value, while the sine function $\sin(\mathcal{L}_f)$ approximates closely to 1. Conversely, when the residual loss value diminishes towards 0, the function $\cos(\mathcal{L}_f)$ converges to 1, and the function $\sin(\mathcal{L}_f)$ diminishes to an infinitesimally small value. To some extent, this way can be equated an adaptive learning rate without the need for additional controls.

According to loss function (Eq. 14), the optimization process of the trainable parameter θ_2 is modified compared to the original L-BFGS optimization algorithm (Liu & Nocedal, 1989). For clarification, here we only show what is different from the original L-BFGS algorithm and take the trainable parameter θ_2 as an example. We firstly define $s_k = \theta_2^{k+1}-\theta_2^k$, the gradient $g_k = \nabla_{\theta_2}\mathcal{L}_f(\theta^k)$ and $y_k = \cos\left(\mathcal{L}_f(\theta^{k+1})\right)g_{k+1}-\cos\left(\mathcal{L}_f(\theta^k)\right)g_k$. Computing

$$d_k = -H_k\cos\left(\mathcal{L}_f(\theta^k)\right)g_k,$$

where

$$H_{k+1} = \left(I-\frac{s_k y_k^T}{y_k^T s_k}\right)H_k\left(I-\frac{y_k s_k^T}{y_k^T s_k}\right)+\frac{s_k s_k^T}{y_k^T s_k}.$$

Then,

$$\theta_2^{k+1} = \theta_2^k+\alpha_k d_k,$$

where α_k satisfies the Wolfe conditions:

$$\mathcal{L}_f\left(\theta^k+\alpha_k d_k\right)\le\mathcal{L}_f\left(\theta^k\right)+\beta'\alpha_k\cos\left(\mathcal{L}_f(\theta^k)\right)g_k^{\mathrm{T}}d_k,$$

$$\cos\left(\mathcal{L}_f(\theta^k+\alpha_k d_k)\right)g\left(\theta^k+\alpha_k d_k\right)^{\mathrm{T}}d_k\ge\beta\cos\left(\mathcal{L}_f(\theta^k)\right)g_k^{\mathrm{T}}d_k.$$

4.3. Experiment

In this part, in order to evaluate the performance of the ResTranPINN method, we choose a high-dimensional, high-order, nonlinear, and time-varying PDE as a numerical example. The training data comes from a solution with a complex evolutionary form, which changes with time, similar to the water wave in the ocean, known as the time-varying feature (Zhang et al., 2021). Therefore, Eq. (5) is suitable for our example, which contains high-order terms and nonlinear terms, where there are multiple parameters and a large scale between the parameters. In the following, we utilize the ResTranPINN method to infer the large-scale parameters. The hyper-parameter choice inferred parameters for the inverse problem that are all the same in each experiment and are presented in Table 7.

Table 7
Hyper-parameter selection for inverse problem.

Hidden layer	2×100
Activation function	Hyperbolic Tangent
Train data	$N_0 = 2k, N_b = 6k, N_f = 20k$
Initialization	Xavier (Glorot & Bengio, 2010)
collocation points sample	Latin Hypercube Sampling (Stein, 1987)
Optimization	L-BFGS (Liu & Nocedal, 1989)

We measured not only the stability of the ResTranPINN method under different seed numbers but also the robustness of the method under different noise data. To evaluate the performance of the ResTranPINN method from multiple perspectives, we present the results for several metrics including running time, number of iterations, and error rate. The error rate is defined as follows,

$$Error\ \% = \frac{|\beta_{pred}-\beta_{GT}|}{|\beta_{GT}|}\times 100\,\%,$$

where β_{pred} represents the inferred value and β_{GT} denotes the GT value. We conduct several experiments in two groups according to the initialization strategy of the inferred parameters. One group initializes the inference parameter θ_2 as zero, and the other one is initialized with uniformly distributed random numbers, namely, $\theta_2\sim U(0,1)$. We also validate separately the robustness of ResTranPINN in each initialization group with or without adding noise, where noise involve Gaussian noise, uniform noise, and impulsive noise. Noise is only added to the initial data such as Gaussian noise $\mathcal{N}(0, Var(u_0(\boldsymbol{x_0})))$.

In Table 8, we present a detailed comparison of the performance indicators for the PINN and ResTranPINN methods in inferring large-scale parameters, considering different initialization strategies and the presence or absence of noise in the data. Overall, the ResTranPINN method significantly outperforms the PINN method in terms of error rate. Specifically, under zero initialization of the inferred parameters, the ResTranPINN method achieves notably higher accuracy compared to the PINN

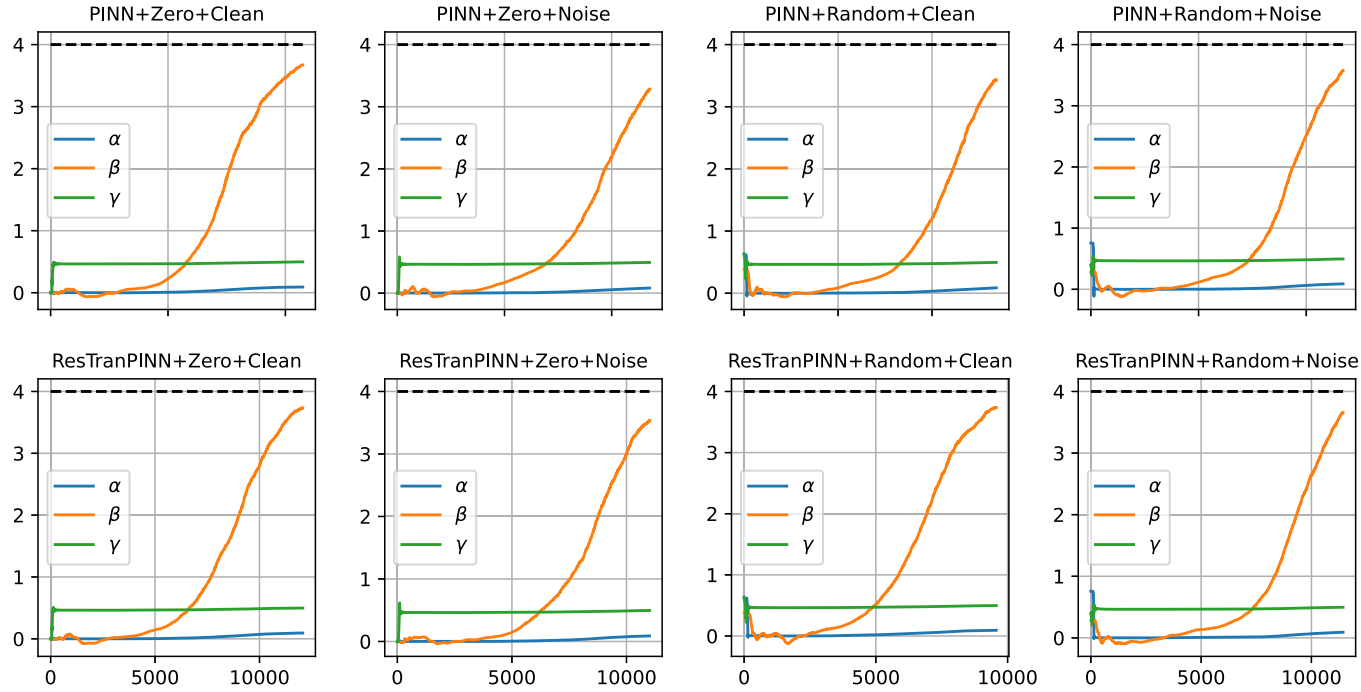


Fig. 8. The PINN and ResTranPINN methods update the records of the inferred parameters under different initialization strategies and training data. Top: The PINN method in **Zero** initialization and **Random** initialization utilizes **Clean** data and **Noise** data for inferred parameters. Bottom: The ResTranPINN method in various initialization and dataset update records for inferred parameters.

Table 8
The error rate and running time and number of iterations for the PINN and ResTranPINN methods for inferring the parameter θ_2 under zero initialization and random initialization with or without adding Gaussian noise. (Time: Seconds, GT values of the parameters α, β and γ are set to $\alpha = 0.1$, $\beta = 4$ and $\gamma = 0.5$ in Ref Zhang et al. (2021), respectively).

Methods	Params.	Index	Zero initialization		Random initialization	
			Clean seed(2)	Noise (10 %) seed(2)	Clean seed(2)	Noise (10 %) seed(1)
PINN	α	Value	0.09166487	0.08193173	0.08608765	0.08893042
		Error %	8.335%	18.068 %	13.912%	11.069 %
	β	Value	3.671845	3.282434	3.4311764	3.5721164
		Error %	8.203%	17.939 %	14.220%	10.697 %
	γ	Value	0.4970383	0.4936206	0.49469197	0.49626827
		Error %	0.592%	1.275 %	1.061%	0.746 %
	Time		434	442	541	475
	Iterations		10,742	11,769	13,418	11,708
ResTranPINN	α	Value	0.0936643	0.0886184	0.09352215	0.09107731
		Error %	**6.335**%	11.381 %	**6.477**%	8.922 %
	β	Value	3.7340884	3.5329244	3.7395296	3.6572237
		Error %	**6.647**%	11.676 %	**6.511**%	8.569 %
	γ	Value	0.49745634	0.49566507	0.49767137	0.49705523
		Error %	**0.508**%	0.866 %	**0.465**%	0.588 %
	Time		488	417	385	462
	Iterations		12,074	11,007	9572	11,434

method, both for clean data and data with 10 % uniform noise. When random initialization is applied, ResTranPINN consistently outperforms PINN across multiple metrics, including lower error rates, reduced computational time, and fewer iterations. These findings underscore the superior stability and robustness of ResTranPINN relative to PINN, particularly in challenging inference scenarios. Notably, the error rate for the parameter γ is significantly lower than that for α and β. We speculate that this discrepancy arises from the increased complexity of inferring α and β, which are likely more sensitive to nonlinearities and high-order terms in the underlying model. Additionally, we present a comprehensive comparison of the error rates, running time, and iteration counts for both the PINN and ResTranPINN methods when inferring the parameter θ_2. This comparison encompasses scenarios with zero initialization and random initialization, both with and without the introduction of uniform noise (10 %) and impulsive noise (1 %). The detailed results are provided in Appendix C, specifically Table C.13.

We show the updated records for inferred parameters during model training using the PINN method and ResTranPINN method in different initialization strategies and training data in Fig. 8. Due to the large-scale phenomenon between parameters, the update of parameters α and γ hardly get any detailed fluctuation information. The black dashed line in each subgraph represents the true value of the parameter β, and we find that the ResTranPINN method is more likely to infer the parameter β.

5. Conclusion and discussion

In this paper, we connect the optimization of PINN with multi-task learning and propose the PINN with a loss-driven dynamic weight, as well as derive its neural tangent kernel theory. We evaluate the effectiveness of the proposed algorithm in combination with different learning rate strategies for complex physical phenomena of high-dimensional equations with high-order terms. The experimental results show that the proposed algorithm is superior to the LRA algorithm generated by the back-propagation gradient in terms of accuracy and time cost. At the same time, we explore the potential advantages of our proposed algorithm, including the allocation principle and update frequency, which can improve the accuracy of the algorithm to varying degrees without sacrificing time costs. We reveal the connection between the optimization of the PINN and the learning rate in terms of time cost and accuracy. However, it is still not excluded that other factors affect the accuracy, including the number of collocation points, the size of the network, the number of iterations, parameters initialization, and so on. In addition to the above forward problem, we develop the PINN theory for the multi-parameter discovery of high-dimensional nonlinear PDEs with high-order terms based on a gradient descent algorithm. We propose a ResTranPINN method, which is applied to multiple parameters with large-scale learnable parameters. We evaluate the robustness of the algorithm in zero initialization and random initialization with or without noise data. The results show that the ResTranPINN algorithm not only improves the accuracy but also is robust on the noisy data. In addition, the experimental results show that strong nonlinearity and high-order terms also affect the accuracy of the inferred parameters.

Despite a fixed update frequency can ensure the consistency of weight adjustment, there are still some accidental phenomena. In addition, the scale factor is a user-defined hyperparameter, which could control instability during training. As a direction for future research, we will propose the development and evaluation of adaptive weight update mechanisms that can dynamically adjust the update interval based on real-time training feedback. There is a vast array of PDEs with different characteristics and applications in various fields such as fluid dynamics, quantum mechanics, and finance. Extending our methods to these other types of PDEs is a logical next step. Integrating our approach with other machine learning algorithms may lead to new insights and improved performance.

CRediT authorship contribution statement

Yabin Zhang: Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Formal analysis, Conceptualization; **Liang-Jian Deng:** Writing – review & editing, Supervision, Project administration, Funding acquisition, Formal analysis.

Data availability statements

The data that support the findings of this study are available from the corresponding author upon reasonable request.

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.

Acknowledgement

This research is supported by the Department of Science and Technology of Sichuan Province (2025YFNH0001) and the National Natural Science Foundation of China (12271083).

Appendix A.

Recall the loss function of the loss-driven dynamic weight PINN method

$$\mathcal{L}(\theta)=\frac{1}{2}\left(\sum_{n=1}^{N_0}|u(\boldsymbol{x}_0^n;\theta)-u_0(\boldsymbol{x}_0^n)|^2+\lambda_b\sum_{n=1}^{N_b}|u(\boldsymbol{x}_b^n;\theta)-u_b(\boldsymbol{x}_b^n)|^2+\sum_{n=1}^{N_f}|f(\boldsymbol{x}_f^n;\theta)|^2\right).$$

By gradient flow

$$\begin{aligned}\frac{d\theta}{dt}&=-\nabla_\theta\mathcal{L}(\theta)\\&=-\Big[\sum_{n=1}^{N_0}\frac{\partial u(\boldsymbol{x}_0^n;\theta(t))}{\partial\theta}(u(\boldsymbol{x}_0^n;\theta(t))-u_0(\boldsymbol{x}_0^n))\\&\quad+\lambda_b\sum_{n=1}^{N_b}\frac{\partial u(\boldsymbol{x}_b^n;\theta(t))}{\partial\theta}(u(\boldsymbol{x}_b^n;\theta(t))-u_b(\boldsymbol{x}_b^n))\\&\quad+\sum_{n=1}^{N_f}\frac{\partial f(\boldsymbol{x}_f^n;\theta(t))}{\partial\theta}f(\boldsymbol{x}_f^n;\theta(t))\Big].\end{aligned}$$

Given the training data $\{\boldsymbol{x}_0^n,u_0(\boldsymbol{x}_0^n)\}_{n=1}^{N_0}$, $\{\boldsymbol{x}_b^n,u_b(\boldsymbol{x}_b^n)\}_{n=1}^{N_b}$, $\{\boldsymbol{x}_f^n\}_{n=1}^{N_f}$, $u(\boldsymbol{x};\theta)$ obeys the following evolution process.

From evolution Eq. (10) for $1\le n\le N_0$,

$$\begin{aligned}\frac{du(\boldsymbol{x}_0^n;\theta(t))}{dt}&=\frac{du(\boldsymbol{x}_0^n;\theta(t))}{d\theta}^T\frac{d\theta}{dt}\\&=-\frac{du(\boldsymbol{x}_0^n;\theta(t))}{d\theta}^T\Big[\sum_{n=1}^{N_0}\frac{\partial u(\boldsymbol{x}_0^n;\theta(t))}{\partial\theta}(u(\boldsymbol{x}_0^n;\theta(t))-u_0(\boldsymbol{x}_0^n))\\&\quad+\lambda_b\sum_{n=1}^{N_b}\frac{\partial u(\boldsymbol{x}_b^n;\theta(t))}{\partial\theta}(u(\boldsymbol{x}_b^n;\theta(t))-u_b(\boldsymbol{x}_b^n))+\sum_{n=1}^{N_f}\frac{\partial f(\boldsymbol{x}_f^n;\theta(t))}{\partial\theta}f(\boldsymbol{x}_f^n;\theta(t))\Big]\\&=-\sum_{n=1}^{N_0}(u(\boldsymbol{x}_0^n;\theta(t))-u_0(\boldsymbol{x}_0^n))\Big\langle\frac{du(\boldsymbol{x}_0^n;\theta(t))}{d\theta},\frac{du(\boldsymbol{x}_0^n;\theta(t))}{d\theta}\Big\rangle\\&\quad-\lambda_b\sum_{n=1}^{N_b}(u(\boldsymbol{x}_b^n;\theta(t))-u_b(\boldsymbol{x}_b^n))\Big\langle\frac{du(\boldsymbol{x}_b^n;\theta(t))}{d\theta},\frac{du(\boldsymbol{x}_0^n;\theta(t))}{d\theta}\Big\rangle\\&\quad-\sum_{n=1}^{N_f}f(\boldsymbol{x}_f^n;\theta(t))\Big\langle\frac{df(\boldsymbol{x}_f^n;\theta(t))}{d\theta},\frac{du(\boldsymbol{x}_0^n;\theta(t))}{d\theta}\Big\rangle.\end{aligned}$$

From evolution Eq. (11) for $1\le n\le N_b$,

$$\begin{aligned}\frac{du(\boldsymbol{x}_b^n,\theta(t))}{dt}&=\frac{du(\boldsymbol{x}_b^n;\theta(t))}{d\theta}^T\frac{d\theta}{dt}\\&=-\frac{du(\boldsymbol{x}_b^n;\theta(t))}{d\theta}^T\Big[\sum_{n=1}^{N_0}\frac{\partial u(\boldsymbol{x}_0^n;\theta(t))}{\partial\theta}(u(\boldsymbol{x}_0^n;\theta(t))-u_0(\boldsymbol{x}_0^n))\\&\quad+\lambda_b\sum_{n=1}^{N_b}\frac{\partial u(\boldsymbol{x}_b^n;\theta(t))}{\partial\theta}(u(\boldsymbol{x}_b^n;\theta(t))-u_b(\boldsymbol{x}_b^n))+\sum_{n=1}^{N_f}\frac{\partial f(\boldsymbol{x}_f^n;\theta(t))}{\partial\theta}f(\boldsymbol{x}_f^n;\theta(t))\Big]\\&=-\sum_{n=1}^{N_0}(u(\boldsymbol{x}_0^n;\theta(t))-u_0(\boldsymbol{x}_0^n))\Big\langle\frac{du(\boldsymbol{x}_0^n;\theta(t))}{d\theta},\frac{du(\boldsymbol{x}_b^n;\theta(t))}{d\theta}\Big\rangle\\&\quad-\lambda_b\sum_{n=1}^{N_b}(u(\boldsymbol{x}_b^n;\theta(t))-u_b(\boldsymbol{x}_b^n))\Big\langle\frac{du(\boldsymbol{x}_b^n;\theta(t))}{d\theta},\frac{du(\boldsymbol{x}_b^n;\theta(t))}{d\theta}\Big\rangle\\&\quad-\sum_{n=1}^{N_f}f(\boldsymbol{x}_f^n;\theta(t))\Big\langle\frac{df(\boldsymbol{x}_f^n;\theta(t))}{d\theta},\frac{du(\boldsymbol{x}_b^n;\theta(t))}{d\theta}\Big\rangle.\end{aligned}$$

Similarly, from evolution Eq. (12) for $1\le n\le N_f$,

$$\begin{aligned}\frac{df(\boldsymbol{x}_f^n;\theta(t))}{dt}&=\frac{df(\boldsymbol{x}_f^n;\theta(t))}{d\theta}^T\frac{d\theta}{dt}\\&=-\frac{df(\boldsymbol{x}_f^n;\theta(t))}{d\theta}^T\Big[\sum_{n=1}^{N_0}\frac{\partial u(\boldsymbol{x}_0^n;\theta(t))}{\partial\theta}(u(\boldsymbol{x}_0^n;\theta(t))-u_0(\boldsymbol{x}_0^n))\\&\quad+\lambda_b\sum_{n=1}^{N_b}\frac{\partial u(\boldsymbol{x}_b^n;\theta(t))}{\partial\theta}(u(\boldsymbol{x}_b^n;\theta(t))-u_b(\boldsymbol{x}_b^n))+\sum_{n=1}^{N_f}\frac{\partial f(\boldsymbol{x}_f^n;\theta(t))}{\partial\theta}f(\boldsymbol{x}_f^n;\theta(t))\Big]\end{aligned}$$

$$= -\sum_{n=1}^{N_0}(u(\boldsymbol{x}_0^n;\theta(t)) - u_0(\boldsymbol{x}_0^n))\Big\langle \frac{du(\boldsymbol{x}_0^n;\theta(t))}{d\theta}, \frac{df(\boldsymbol{x}_f^n;\theta(t))}{d\theta}\Big\rangle$$
$$- \lambda_b \sum_{n=1}^{N_b}(u(\boldsymbol{x}_b^n;\theta(t)) - u_b(\boldsymbol{x}_b^n))\Big\langle \frac{du(\boldsymbol{x}_b^n;\theta(t))}{d\theta}, \frac{df(\boldsymbol{x}_f^n;\theta(t))}{d\theta}\Big\rangle$$
$$- \sum_{n=1}^{N_f} f(\boldsymbol{x}_f^n;\theta(t))\Big\langle \frac{df(\boldsymbol{x}_f^n;\theta(t))}{d\theta}, \frac{df(\boldsymbol{x}_f^n;\theta(t))}{d\theta}\Big\rangle.$$

Then we can rewrite the above equations as

$$\begin{aligned}
\frac{du(\boldsymbol{x}_0;\theta(t))}{dt} =& -\boldsymbol{K}_0(t)\cdot(u(\boldsymbol{x}_0;\theta(t)) - u_0(\boldsymbol{x}_0))\\
&- \lambda_b \boldsymbol{K}_{0b}(t)\cdot(u(\boldsymbol{x}_b;\theta(t)) - u_b(\boldsymbol{x}_b))\\
&- \boldsymbol{K}_{0f}(t)\cdot f(\boldsymbol{x}_f;\theta(t)),\\
\frac{du(\boldsymbol{x}_b;\theta(t))}{dt} =& -\boldsymbol{K}_{b0}(t)\cdot(u(\boldsymbol{x}_0;\theta(t)) - u_0(\boldsymbol{x}_0))\\
&- \lambda_b \boldsymbol{K}_b(t)\cdot(u(\boldsymbol{x}_b;\theta(t)) - u_b(\boldsymbol{x}_b))\\
&- \boldsymbol{K}_{bf}(t)\cdot f(\boldsymbol{x}_f;\theta(t)),\\
\frac{df(\boldsymbol{x}_f;\theta(t))}{dt} =& -\boldsymbol{K}_{f0}(t)\cdot(u(\boldsymbol{x}_0;\theta(t)) - u_0(\boldsymbol{x}_0))\\
&- \lambda_b \boldsymbol{K}_{fb}(t)\cdot(u(\boldsymbol{x}_b;\theta(t)) - u_b(\boldsymbol{x}_b))\\
&- \boldsymbol{K}_f(t)\cdot f(\boldsymbol{x}_f;\theta(t)),
\end{aligned}$$

Appendix B.

where $\boldsymbol{K}_{0b}(t) = \boldsymbol{K}_{b0}^{\mathrm{T}}(t) \in \mathbb{R}^{N_0\times N_b}$, $\boldsymbol{K}_{0f}(t) = \boldsymbol{K}_{f0}^{\mathrm{T}}(t) \in \mathbb{R}^{N_0\times N_f}$, $\boldsymbol{K}_{bf}(t) = \boldsymbol{K}_{fb}^{\mathrm{T}}(t) \in \mathbb{R}^{N_b\times N_f}$, $\boldsymbol{K}_0(t) \in \mathbb{R}^{N_0\times N_0}$, $\boldsymbol{K}_b(t) \in \mathbb{R}^{N_b\times N_b}$, $\boldsymbol{K}_f(t) \in \mathbb{R}^{N_f\times N_f}$,

$$\begin{aligned}
(\boldsymbol{K}_0)_{ij}(t) &= \Big\langle \frac{du(\boldsymbol{x}_0^i;\theta_t)}{d\theta}, \frac{du(\boldsymbol{x}_0^j;\theta_t)}{d\theta}\Big\rangle, (\boldsymbol{K}_b)_{ij}(t)\\
&= \Big\langle \frac{du(\boldsymbol{x}_b^i;\theta_t)}{d\theta}, \frac{du(\boldsymbol{x}_b^j;\theta_t)}{d\theta}\Big\rangle,\\
(\boldsymbol{K}_f)_{ij}(t) &= \Big\langle \frac{df(\boldsymbol{x}_f^i;\theta_t)}{d\theta}, \frac{df(\boldsymbol{x}_f^j;\theta_t)}{d\theta}\Big\rangle, (\boldsymbol{K}_{0b})_{ij}(t)\\
&= \Big\langle \frac{du(\boldsymbol{x}_0^i;\theta_t)}{d\theta}, \frac{du(\boldsymbol{x}_b^j;\theta_t)}{d\theta}\Big\rangle,\\
(\boldsymbol{K}_{0f})_{ij}(t) &= \Big\langle \frac{du(\boldsymbol{x}_0^i;\theta_t)}{d\theta}, \frac{df(\boldsymbol{x}_f^j;\theta_t)}{d\theta}\Big\rangle, (\boldsymbol{K}_{bf})_{ij}(t)\\
&= \Big\langle \frac{du(\boldsymbol{x}_b^i;\theta_t)}{d\theta}, \frac{df(\boldsymbol{x}_f^j;\theta_t)}{d\theta}\Big\rangle.
\end{aligned}$$

Table B.9
The relative L_2 error and running time for case 1 based on different allocation principles (S) and update frequencies (P) with constant learning rate. (Time: Seconds).

		$P = 1$	$P = 20$	$P = 30$	$P = 40$	$P = 50$
$S = 1000$	Rel. L_2	**3.0832e-02**	3.4961e-02	3.5431e-02	3.6525e-02	3.9401e-02
	Time	596.702	584.659	584.346	584.537	585.442
$S = 2000$	Rel. L_2	3.7152e-02	4.0174e-02	4.2599e-02	3.9414e-02	3.8832e-02
	Time	597.784	586.527	586.386	586.261	586.147
$S = 3000$	Rel. L_2	5.3515e-02	4.5415e-02	4.3656e-02	3.8470e-02	3.6325e-02
	Time	595.340	584.654	584.420	585.354	584.627
$S = 4000$	Rel. L_2	4.1018e-02	4.4799e-02	3.9711e-02	3.7657e-02	3.9331e-02
	Time	597.545	586.728	588.917	587.808	586.849
$S = 5000$	Rel. L_2	4.1483e-02	3.3127e-02	3.4952e-02	3.4384e-02	3.5021e-02
	Time	595.668	584.602	585.240	584.756	585.151

Table B.10
The relative L_2 error and running time for case 1 based on different allocation principles (S) and update frequencies (P) with piece-wise constant learning rate. (Time: Seconds).

		$P = 1$	$P = 20$	$P = 30$	$P = 40$	$P = 50$
$S = 1000$	Rel. L_2	3.3839e-02	2.8412e-02	2.9771e-02	2.8044e-02	2.7979e-02
	Time	596.044	584.539	584.447	584.485	584.532
$S = 2000$	Rel. L_2	2.5961e-02	2.0267e-02	**1.8874e-02**	2.0353e-02	2.0280e-02
	Time	595.598	584.551	584.438	584.419	584.501
$S = 3000$	Rel. L_2	5.0157e-02	1.9489e-02	1.9544e-02	1.9745e-02	1.9781e-02
	Time	595.215	584.627	584.920	584.544	584.889
$S = 4000$	Rel. L_2	2.8032e-02	2.4221e-02	2.4410e-02	2.4277e-02	2.4298e-02
	Time	595.832	584.419	584.385	584.295	584.117
$S = 5000$	Rel. L_2	2.3908e-02	2.2869e-02	2.2679e-02	2.2688e-02	2.2631e-02
	Time	595.759	584.581	584.304	584.499	584.471

Table B.11
The relative L_2 error and running time for case 1 based on different allocation principles (S) and update frequencies (P) with exponential learning rate. (Time: Seconds).

		$P = 1$	$P = 20$	$P = 30$	$P = 40$	$P = 50$
$S = 1000$	Rel. L_2	8.2487e-02	4.9063e-02	4.2897e-02	4.0735e-02	4.0384e-02
	Time	595.883	584.961	584.700	583.954	584.249
$S = 2000$	Rel. L_2	6.5265e-02	4.1945e-02	4.2201e-02	4.1497e-02	4.1172e-02
	Time	595.687	584.871	584.424	584.808	584.458
$S = 3000$	Rel. L_2	5.6303e-02	3.9390e-02	**3.9348e-02**	3.9520e-02	3.9521e-02
	Time	595.972	585.099	584.463	584.390	584.760
$S = 4000$	Rel. L_2	7.2457e-02	4.8651e-02	5.3048e-02	4.9287e-02	4.9555e-02
	Time	595.796	584.579	584.628	584.672	584.409
$S = 5000$	Rel. L_2	6.9783e-02	6.6293e-02	6.5946e-02	6.6164e-02	6.6083e-02
	Time	595.853	584.571	584.386	584.423	584.310

Table B.12
The relative L_2 error and running time for case 1 based on different allocation principles (S) and update frequencies (P) with cosine learning rate. (Time: Seconds).

		$P = 1$	$P = 20$	$P = 30$	$P = 40$	$P = 50$
$S = 1000$	Rel. L_2	5.9705e-02	3.4278e-02	2.9217e-02	3.4426e-02	3.4599e-02
	Time	595.818	585.213	584.644	584.282	584.758
$S = 2000$	Rel. L_2	3.1593e-02	3.2384e-02	3.1855e-02	2.5525e-02	2.4862e-02
	Time	595.679	584.700	584.426	584.377	584.350
$S = 3000$	Rel. L_2	2.1129e-02	1.8811e-02	8.8565e-02	**1.8403e-02**	1.8524e-02
	Time	595.894	584.915	584.392	584.370	584.540
$S = 4000$	Rel. L_2	2.6585e-02	2.1450e-02	2.1496e-02	6.9906e-02	2.1889e-02
	Time	595.993	584.883	584.509	584.402	584.301
$S = 5000$	Rel. L_2	2.4143e-02	1.9940e-02	2.0326e-02	2.0430e-02	2.0591e-02
	Time	595.943	593.198	586.754	608.278	594.232

Appendix C.

Table C.13
The error rate and running time and number of iterations for the PINN and ResTranPINN methods for inferring the parameter θ_2 under zero initialization and random initialization with or without adding uniform noise(10 %) and impulsive noise(1 %). (Time: Seconds, GT values of the parameters α, β and γ are set to $\alpha = 0.1$, $\beta = 4$ and $\gamma = 0.5$ in Ref. Zhang et al. (2021), respectively).

Methods	Params.	Index	Zero initialization		Random initialization	
			uniform seed(1)	impulsive seed(8)	uniform seed(1)	impulsive seed(10)
PINN	α	Value	0.08611635	0.07251121	0.08590958	0.09434358
		Error %	13.883%	27.488 %	14.090%	5.656 %
	β	Value	3.4541967	2.9159794	3.4371898	3.7591834
		Error %	13.645%	27.100 %	14.070%	6.020 %
	γ	Value	0.4951385	0.49031785	0.4949636	0.4976924
		Error %	0.972%	1.936 %	1.007%	0.461 %
	Time		465	405	453	406
	Iterations		11,212	9718	10,928	9722
ResTranPINN	α	Value	0.08612619	0.07996956	0.09265709	0.09520032
		Error %	13.873%	20.030 %	7.342%	4.799 %
	β	Value	3.4421723	3.2090204	3.7170205	3.79961
		Error %	13.945%	19.774 %	7.074%	5.009 %
	γ	Value	0.49495324	0.4928688	0.4975255	0.49817082
		Error %	1.009%	1.426 %	0.494%	0.365 %
	Time		552	389	379	504
	Iterations		13,429	9411	9205	12,097

References

Ablowitz, M. J., & Segur, H. (1979). On the evolution of packets of water waves. *Journal of Fluid Mechanics*, ***92***, 691–715.

Ali, A., Ullah, I., Ahmad, S., Wu, Z., Li, J., & Bai, X. (2025a). An attention-driven spatio-temporal deep hybrid neural networks for traffic flow prediction in transportation systems. *IEEE Transactions on Intelligent Transportation Systems*, (pp. 1–15).

Ali, A., Ullah, I., Shabaz, M., Sharafian, A., Attique Khan, M., Bai, X., & Qiu, L. (2024). A resource-aware multi-graph neural network for urban traffic flow prediction in multi-access edge computing systems. *IEEE Transactions on Consumer Electronics*, *70*, 7252–7265.

Ali, A., Ullah, I., Singh, S. K., Sharafian, A., Jiang, W., Hammad, I. S., & Bai, X. (2025b). Energy-efficient resource allocation for urban traffic flow prediction in edge-cloud computing. *International Journal of Intelligent Systems*, *2025*, 1863025.

Baker, N., Alexander, F., Bremer, T., Hagberg, A., Kevrekidis, Y., Najm, H., Parashar, M., Patra, A., Sethian, J., Wild, S., Willcox, K., & Lee, S. (2019). Workshop report on basic research needs for scientific machine learning: Core technologies for artificial intelligence.

Baydin , A. G., Barak , A. P., Andreyevich , A. R. & Siskind , J. M. (2018). Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research: JMLR*, *18*, 1–43.

Bischof, R., & Kraus, M. A. (2025). Multi-Objective loss balancing for physics-informed deep learning. *Computer Methods in Applied Mechanics and Engineering*, *439*, 117914.

Dashtbayaz, N. H., Farhani, G., Wang, B., & Ling, C. X. (2024). Physics-informed neural networks: Minimizing residual loss with wide networks and effective activations. IJCAI (pp. 5853–5861).

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research: JMLR*, *12*, 2121–2159.

Farhani, G., Dashtbayaz, N. H., Kazachek, A., & Wang, B. (2025). A simple remedy for failure modes in physics informed neural networks. *Neural Networks*, *183*, 106963.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249–256). PMLR (*vol. 9*).

Jacot, A., Gabriel, F., & Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*. (*vol. 31*).

Jagtap, A. D., Kharazmi, E., & Karniadakis, G. E. (2020). Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, *365*, 113028.

Kadomtsev, B. B., & Petviashvili, V. I. (1970). On the stability of solitary waves in weakly dispersing media. *Soviet Physics-Doklady*, *15*, 539.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*, 436–444.

Lin, S., & Chen, Y. (2022). A two-stage physics-informed neural network method based on conserved quantities and applications in localized wave solutions. *Journal of Computational Physics*, *457*, 111053.

Liu, D. C., & Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, *45*, 503-528.

Lu, L., Meng, X., Mao, Z., & Karniadakis, G. E. (2021). DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, *63*, 208–228.

Ma, W. X. (2011). Comment on the 3+1 dimensional kadomtsev-petviashvili equations. *Communications in Nonlinear Science and Numerical Simulation*, *16*, 2663–2666.

McClenny, L. D., & Braga-Neto, U. M. (2023). Self-adaptive physics-informed neural networks. *Journal of Computational Physics*, *474*, 111722.

Pang, G., Lu, L., & Karniadakis, G. E. (2019). FPINNs: Fractional physics-informed neural networks. *SIAM Journal of Scientific Computing*, *41*, A2603–A2626.

Qian, C., Rao, J., Liu, Y., & He, J. (2016). Rogue waves in the three-dimensional kadomtsev-petviashvili equation. *Chinese Physics Letters*, *33*, 110201.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, *378*, 686–707.

Senatorski, A., & Infeld, E. (1996). Simulations of two-dimensional kadomtsev-petviashvili soliton dynamics in three-dimensional space. *Physical Review Letters*, *77*, 2855–2858.

Song, Y., Wang, H., Yang, H., Taccari, M. L., & Chen, X. (2024). Loss-attentional physics-informed neural networks. *Journal of Computational Physics*, *501*, 112781.

Stein, M. (1987). Large sample properties of simulations using latin hypercube sampling. *Technometrics: A Journal of Statistics for the Physical, Chemical, and Engineering Sciences*, *29*, 143–151.

Tieleman, T., Hinton, G. et al. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, *4*, 26–31.

Ullah, I., Adhikari, D., Ali, F., Ali, A., Khan, H., Sharafian, A., Manic Kesavan, S., & Bai, X. (2024). Revolutionizing E-commerce with consumer-driven energy-efficient WSNs: A multi-characteristics approach. *IEEE Transactions on Consumer Electronics*, *70*, 6871–6882.

Wang, H., Lu, L., Song, S., & Huang, G. (2023). Learning specialized activation functions for physics-informed neural networks. *Communications in Computational Physics*, *34*, 869–906.

Wang, S., Sankaran, S., & Perdikaris, P. (2024). Respecting causality for training physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, *421*, 116813.

Wang, S., Teng, Y., & Perdikaris, P. (2021). Understanding and mitigating gradient flow pathologies in physics-Informed neural networks. *SIAM Journal of Scientific Computing*, *43*, A3055–A3081.

Wang, S., Yu, X., & Perdikaris, P. (2022). When and why PINNs fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, *449*, 110768.

Wu, C., Zhu, M., Tan, Q., Kartha, Y., & Lu, L. (2023). A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. *Comput. Methods Appl. Mech. Engrg. 403*, 115671.

Xiang, Z., Peng, W., Liu, X., & Yao, W. (2022). Self-adaptive loss balanced physics-informed neural networks. *Neurocomputing*, *496*, 11–34.

Yang, L., Zhang, D., & Karniadakis, G. E. (2020). Physics-informed generative adversarial networks for stochastic differential equations. *SIAM Journal of Scientific Computing*, *42*, A292–A317.

Yuan, L., Ni, Y. Q., Deng, X. Y., & Hao, S. (2022). A-PINN: Auxiliary physics informed neural networks for forward and inverse problems of nonlinear integro-differential equations. *Journal of Computational Physics*, *462*, 111260.

Zhang, D., Guo, L., & Karniadakis, G. E. (2020). Learning in modal space: Solving time-Dependent stochastic PDEs using physics-informed neural networks. *SIAM Journal of Scientific Computing*, *42*, A639–A665.

Zhang, D., Lu, L., Guo, L., & Karniadakis, G. E. (2019). Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *Journal of Computational Physics*, *397*, 108850.

Zhang, D., Wang, L., Liu, L., Liu, T., & Sun, W. (2021). Shape-changed propagations and interactions for the (3+1)-dimensional generalized kadomtsev-petviashvili equation in fluids. *Communications in Theoretical Physics*, *73*, 095001.

Zhu, W., Khademi, W., Charalampidis, E. G., & Kevrekidis, P. G. (2022). Neural networks enforcing physical symmetries in nonlinear dynamical lattices: The case example of the ablowitz–Ladik model. *Physica D: Nonlinear Phenomena*, *434*, 133264.