Electronic Research Archive

https://www.aimspress.com/journal/era

ERA, 34(1): 113–159.
DOI: 10.3934/era.2026007
Received: 10 October 2025
Revised: 17 December 2025
Accepted: 22 December 2025
Published: 30 December 2025

Research article

The improved PINNs for solving partial differential equations with nonlinear couple systems

Jie Xu[1], **Jinhua Ran**[1,2,*] **and Jiao She**[1]

[1] School of Mathematics and Information Science, North Minzu University, Yinchuan 750021, China
[2] Ningxia Key Laboratory of Intelligent Information and Big Data Processing, Yinchuan 750021, China

* **Correspondence:** Email: jhran@nmu.edu.cn.

Abstract: For partial differential equations, traditional methods are characterized by high accuracy, yet they face difficulties in handling complex high-dimensional. Although physics-informed neural networks (PINNs) can effectively solve high-dimensional problems, their accuracy still needs to be improved. The Fourier neural operator (FNO) has an advantage in capturing the global characteristics of physical systems, but it relies on a large amount of training data. To address these limitations, this paper proposes a hybrid method, FNO-PINN, which integrates the FNO with a PINN. This method devises a hybrid architecture consisting of an enhanced FNO, high precision difference calculation, multi-objective physical constraints, and post processing mechanisms. Meanwhile, a dynamic weight adjustment strategy and an iterative auxiliary data loss for different time-scale variables are proposed to effectively alleviate the imbalance problem during the training process. The numerical experimental results of three examples show that compared with the traditional PINNs method, the proposed FNO-PINN method has a faster convergence speed and smaller error, and it demonstrates superior performance in handling coupled systems with multiple time scales.

Keywords: finite difference method; loss function; Fourier neural operator; physics-informed neural networks

1. Introduction

Partial differential equations (PDEs) can be employed to describe complex natural phenomena and are extensively utilized in different fields, such as chemistry [1], biology [2], and physics [3]. However, most complex PDEs cannot be solved directly, and the numerical solutions can be obtained by various methods, including the finite difference method (FDM) [4], the finite element method [5], the integral factor method [6], the spectral method [7], physical information neural networks (PINNs) [8], and so

on. In these methods, although the traditional method represented by FDM has the characteristics of simple implementation, high numerical precision, and small error [9], there are some challenges when dealing with complex problems in high dimensions [10].

In recent years, with the rapid development of deep learning and neural networks, PINNs, a novel method for solving PDEs, has garnered increasing attention. First proposed by Raissi et al. [8], this method generates an approximate solution to PDEs by training a neural network to minimize a predefined loss function. Beck et al. [11] proposed approximation algorithms based on deep learning which may be more efficient than standard Monte Carlo (MC) methods. An auxiliary physical information neural network (A-PINN) framework was introduced to solve the forward and inverse problems of nonlinear equations [12]. PINNs were used to obtain an approximate solution to the inverse problem [13]. As the complexity and dimension of the system increases, solved PDEs by classical PINNs [14] can produce the curse of dimension. Therefore, in order to get the better results, it is necessary to optimize the performance of PINNs [15–17]. Segmentation and parallel training were carried out on the time and space domains in [18]. Guo et al. [19] introduced a MC-PINN method, which provides an effective method for solving high-dimensional PDEs.

In conclusion, although PINNs are powerful in dealing with the PDEs, some limitations lie in their limited accuracy.

As a recently developed operator learning method, the Fourier neural operator (FNO) was proposed by Li et al. [20], which is an innovative method for learning operator mappings in function spaces. The core idea of an FNO is to transform the problem into the frequency domain using the Fourier transform, learn a parameterized linear transformation in the frequency domain, and then return to the physical space through the inverse transform. FNOs show excellent performance in a variety of PDE solving tasks, including Navier-Stokes equations [20], and more. These studies demonstrate the powerful ability of FNOs to capture the global dynamics of complex physical systems, but most of them rely on large amounts of training data and lack direct consideration of physical constraints.

In recent years, hybrid models that combine the advantages of multiple methods have shown great potential for solving the PDEs [21]. For PINNs, several studies have attempted to improve their performance by introducing auxiliary information or combining other methods. Wang et al. [15] proposed that by designing Fourier feature mapping to adjust the spectral bias of PINNs, the problem of their handling of multi-scale PDEs was effectively solved. Cai et al. [22] proposed a PINN method that integrates the heat transfer equation with sparse temperature data to solve complex inverse problems such as unknown thermal boundaries. However, the hybrid model studies specifically addressing coupled systems at multiple time scales are still limited. A framework integrating long short-term memory networks and feed-forward neural networks was proposed [23], which enabled precise prediction of complex path-dependent behaviors and physically interpretable modeling.

In summary, the FDM has high accuracy in solving PDEs, but it has limitations when dealing with high-dimensional and complex problems. PINNs can effectively solve high-dimensional complex PDEs, but the accuracy needs to be improved. FNOs are powerful in capturing the global dynamics of complex physical systems, but mostly rely on large amounts of training data and lack direct consideration of physical constraints. To address these issues, this paper proposes an innovative FNO-PINN hybrid model framework for the coupled equation system. Our method integrates the frequency domain learning capability of FNOs and the physical constraint mechanism of PINNs, which will be supplemented by a time-adaptive weight strategy and multi-objective physical

constraints, to successfully solve the accuracy and stability challenges. The main contributions are as follows: 1) A hybrid FNO-PINN architecture is designed to make full use of the learning ability of FNOs in the frequency domain and the physical constraints of PINNs. 2) A dynamic weight-adjustment strategy variables is proposed to alleviate the imbalance problem in the training process. 3) A multi-objective loss function and post-processing mechanism are developed to ensure that the solution satisfies the PDEs, boundary conditions, and initial conditions simultaneously. 4) A progressive auxiliary data loss is introduced to improve training efficiency and stability while maintaining physical constraints.

The structure of this paper is as follows: Section 2 presents problem reformulation and prerequisites. The FNO-PINN is proposed in Section 3. In Section 4, the performance of FNO-PINN and PINNs in solving PDEs is compared through numerical experiments. Section 5 summarizes the results and looks forward to future research directions.

2. Problem reformulation and prerequisites

2.1. Physics-informed neural networks

The general form of PDEs can be represented within the framework of PINNs:

$$\mathcal{N}(u;\lambda) = 0, \quad (x,t) \in \Omega \times [0,T], \tag{2.1}$$

where $\mathcal{N}(u;\lambda)$ represents a partial differential operator composed of the function u to be solved and its derivatives the Laplacian operator Δu, with λ being the known physical parameters. (x,t) denotes the space-time coordinates defined over the domain Ω and the time interval $[0,T]$. Typically, PDEs are required to satisfy certain initial and boundary conditions. The initial condition is given by $u(x,0) = u_0(x)$ for $x \in \Omega$, and the boundary condition is expressed as $u(x,t) = g(x,t)$ for $(x,t) \in \partial\Omega \times [0,T]$.

The objective of PINNs is to construct a neural network such that the output of the network, $u_\theta(x,t)$, approximates the solution $u(x,t)$ of the PDEs. The neural network can be expressed as a composition of layers:

$$u_\theta(x,t) = \mathcal{N}_L \circ \sigma \circ \mathcal{N}_{L-1} \circ \cdots \circ \sigma \circ \mathcal{N}_1(x,t), \tag{2.2}$$

where L is the number of layers, $\mathcal{N}_l(\mathbf{z}) = W_l\mathbf{z} + b_l$ represents the affine transformation at layer l with weight matrix $W_l \in \mathbb{R}^{d_l \times d_{l-1}}$ and bias vector $b_l \in \mathbb{R}^{d_l}$, and σ is the nonlinear activation function. In this work, we employ the hyperbolic tangent activation function:

$$\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.3}$$

The architecture of PINNs is described in Figure 1.

To train this neural network, we have to structure its loss function to incorporate multiple physical constraints. The distance-based loss function quantifies the difference between the actual value of the sample and the predicted value of the model in the feature space. The reduction of the distance between two points in the feature space means the improvement of the prediction ability of the model. The mean squared error (MSE) optimizes the fit of the sample to the regression curve by minimizing the squared loss. A low MSE value indicates that the prediction model describes the sample data more accurately.

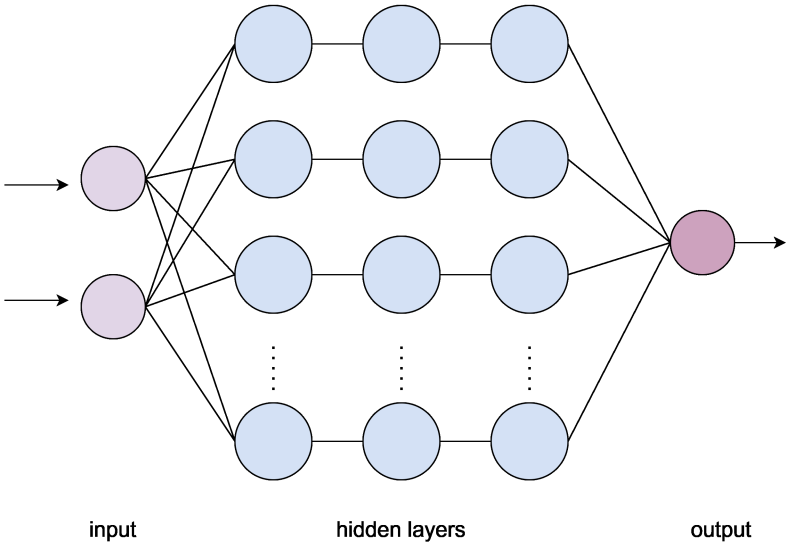


Figure 1. PINNs architecture diadram.

The loss function of PINNs comprises three components. The loss term associated with initial conditions (IC Loss) is

$$\mathcal{L}_{ic} = \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} |u_\theta(x_i, 0) - u_0(x_i)|^2. \tag{2.4}$$

This loss term is used to measure the deviation of the neural network u_θ from the known initial condition $u_0(x)$ at the initial time $t = 0$.

The boundary condition loss term (BC Loss) is shown as follows:

$$\mathcal{L}_{bc} = \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} |u_\theta(x_i, t_i) - g(x_i, t_i)|^2. \tag{2.5}$$

It is used to quantify the deviation of the neural network u_θ from the known boundary condition $g(x, t)$ on the spatial boundary $\partial\Omega$.

The physical equation loss term (PDEs Loss) is expressed by

$$\mathcal{L}_{PDEs} = \frac{1}{N_{PDEs}} \sum_{i=1}^{N_{PDEs}} |\mathcal{N}(u_\theta(x_i, t_i); \lambda)|^2 . \tag{2.6}$$

This loss term is used to measure whether the neural network u_θ satisfies the physical equation $\mathcal{N}(u; \lambda) = 0$ at internal collocation points.

The total loss term (Total Loss) is

$$\mathcal{L} = \alpha_{ic}\mathcal{L}_{ic} + \alpha_{bc}\mathcal{L}_{bc} + \alpha_{PDEs}\mathcal{L}_{PDEs}, \tag{2.7}$$

where $\alpha_{ic}, \alpha_{bc}, \alpha_{PDEs}$ are the weights used to adjust the importance of each loss term.

The loss function $\mathcal{L}$ is minimized as the objective function, which constructs the following optimization problem:

$$\min_{\theta} \mathcal{L}(\theta). \tag{2.8}$$

The neural network parameters θ are optimized using a gradient descent method, such as the Adam optimizer. It adaptively adjusts the learning rate and exhibits excellent performance in most cases. Using forward propagation, input sample points (x, t) are processed to compute the neural network output $u_\theta(x, t)$. The derivatives of u_θ with respect to x and t are calculated using finite difference schemes and substituted into the PDEs. This allows for the computation of the loss function. The gradients of the loss function with respect to the network parameters are then calculated using back propagation, and the network parameters θ are updated using the Adam optimization algorithm.

2.2. Neural operators and Fourier neural operators

Neural operators are a generalization of standard deep neural networks in the operator setting and are used to learn mappings between function spaces for a given PDE and its corresponding solution operator $G^\dagger$, which can be approximated using the neural operator G_θ with parameters θ as a surrogate model.

Neural operators approximate highly nonlinear operators by combining a linear integral operator K with a pointwise nonlinear activation function σ;

$$G_\theta = Q \circ (W_L + K_L) \circ \cdots \circ \sigma(W_1 + K_1) \circ P, \tag{2.9}$$

where P and Q are pointwise operators, W_l is a pointwise linear operator, and K_l is an integral core operator ($l = 1, ..., L$).

The FNO restricts the integral operator K to a convolutional form and is efficiently computed using the fast Fourier transform (FFT) as follows:

$$(Kv_l)(x) = \mathcal{F}^{-1}\left[R \cdot (\mathcal{F}v_l)\right](x) \quad \forall x \in D, \tag{2.10}$$

where $\mathcal{F}, \mathcal{F}^{-1}$ are FFT and its inverse transform, respectively, and R is part of the parameter θ.

Neural operator learning methods are similar to supervised learning in computer vision and language processing, and their performance is highly dependent on the training data. FNO models trained on one set of parameter conditions which cannot easily generalize to other conditions, and it is difficult to collect representative datasets for complex PDEs whose solvers are slow to compute or do not exist.

3. FNO-PINN

We propose the FNO-PINN hybrid model, which organically integrates the frequency domain expression ability of FNOs and the physical constraint mechanism of PINNs. FNO-PINN contains four core modules: enhanced FNO, high-precision differential calculation, multi-objective physical constraints, and post-processing mechanism. The framework is shown in Figure 2 .

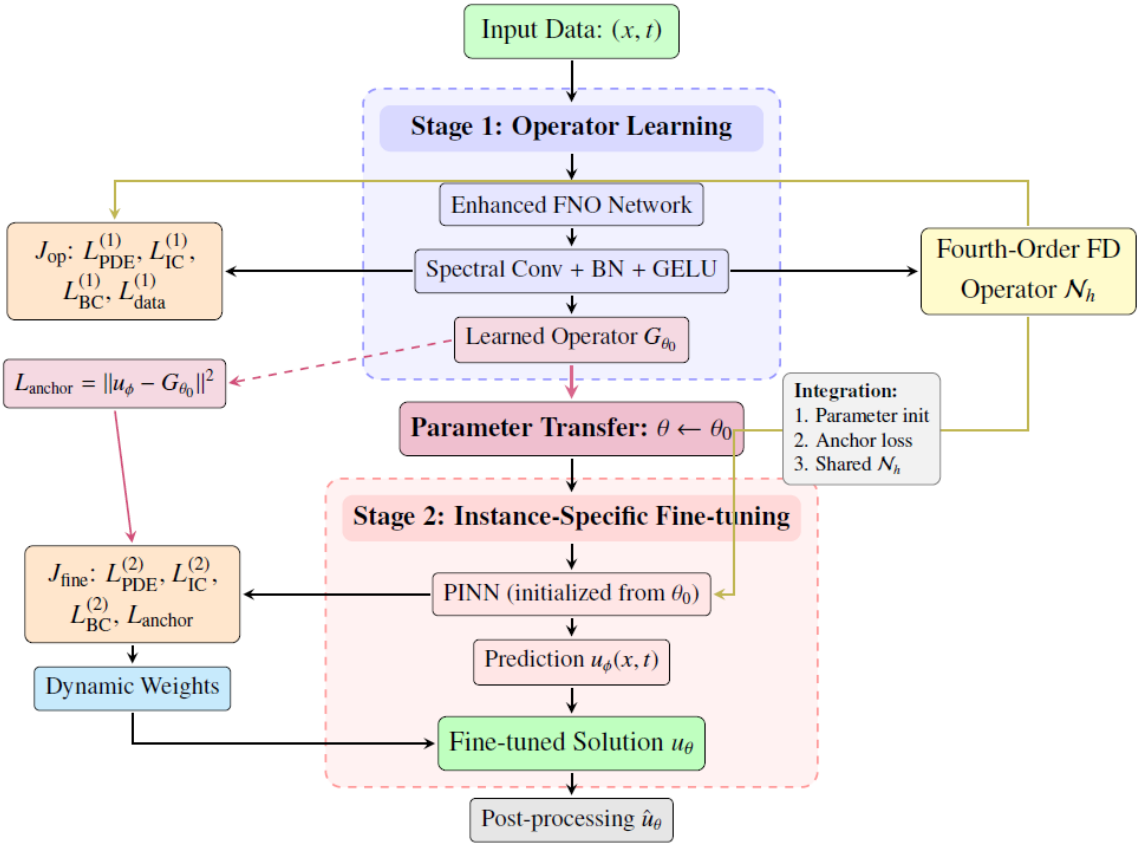


Figure 2. FNO-PINN hybrid architecture.

The core idea of FNO-PINN is to capture the global characteristics of the solution by using the efficient learning ability of FNOs in the frequency domain, while ensuring that the solution satisfies the underlying equations and boundary conditions through the physical constraint mechanism of PINNs. Compared with traditional methods, FNO-PINN has unique advantages in efficiently processing complex nonlinear and multi-scale features through frequency domain learning. Adaptive weighting strategies are designed for variables with different time scales. A high-precision difference scheme is used to improve the accuracy of partial derivative calculation. A two-stage training strategy is introduced to balance the generalization ability and the accuracy of the solution.

Specifically, the FNO and PINN components in our framework are tightly integrated rather than operating in parallel. This integration is achieved through three key mechanisms. Parameter transfer: the PINN in Stage 2 starts with the parameters θ_0 from Stage 1 FNO, so it keeps all the frequency-domain features. Anchor loss: L_{anchor} keeps the Stage 2 solution close to the Stage 1 result. This stops the solution from changing too much. Shared discretization: both stages employ the same fourth-order finite difference operator $\mathcal{N}_h$ to compute PDE residuals, ensuring that physical constraints are enforced in a unified manner.

3.1. Enhanced FNO

The enhanced FNO is shown in Figure 3 .

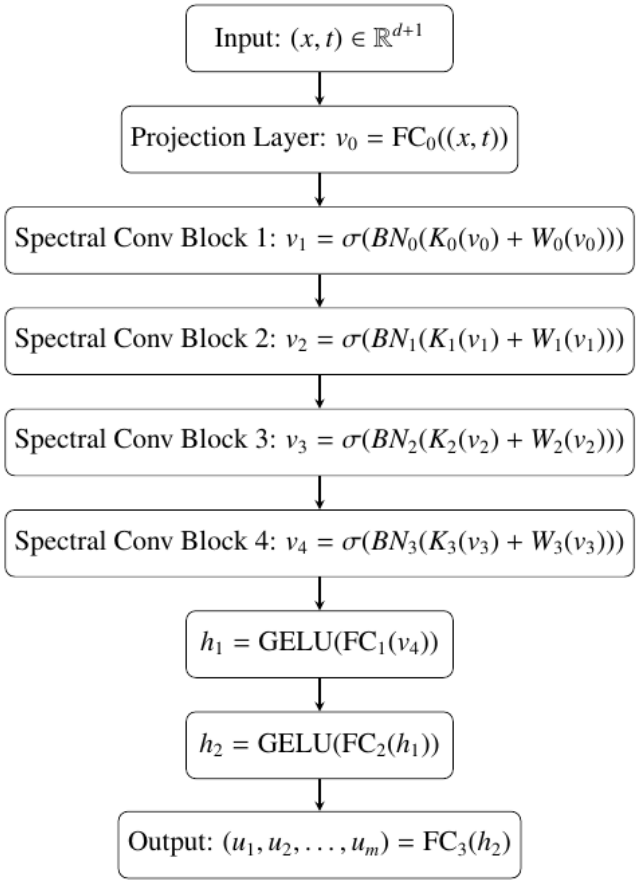


Figure 3. Enhanced FNO architecture.

3.1.1. Frequency domain transform and convolution

The core idea of the FNO is to convert the traditional spatial convolution into the frequency domain multiplication operation, which greatly improves computational efficiency. For a square integrable function $v \in L^2(\Omega)$ defined on a bounded region $\Omega \subset \mathbb{R}^d$, the Fourier transform $\mathcal{F}$ and the inverse transform $\mathcal{F}^{-1}$ are defined as follows:

$$\hat{v}(\boldsymbol{k}) = \mathcal{F}[v](\boldsymbol{k}) = \int_{\Omega} v(\boldsymbol{x}) e^{-i\boldsymbol{k}\cdot\boldsymbol{x}} d\boldsymbol{x}, \quad \boldsymbol{k} \in \mathbb{R}^d \tag{3.1}$$

$$v(\boldsymbol{x}) = \mathcal{F}^{-1}[\hat{v}](\boldsymbol{x}) = \frac{1}{(2\pi)^d} \int_{\mathbb{R}^d} \hat{v}(\boldsymbol{k}) e^{i\boldsymbol{k}\cdot\boldsymbol{x}} d\boldsymbol{k}, \quad \boldsymbol{x} \in \Omega. \tag{3.2}$$

According to the convolution theorem, the convolution operation in the spatial domain is equivalent to the dot multiplication operation in the frequency domain.

$$(v_1 * v_2)(\boldsymbol{x}) = \mathcal{F}^{-1}[\mathcal{F}[v_1] \cdot \mathcal{F}[v_2]](\boldsymbol{x}) \tag{3.3}$$

Based on this theorem, we define the frequency domain convolution operation in the FNO as

$$(\mathcal{K}v)(\boldsymbol{x}) = \mathcal{F}^{-1}[\boldsymbol{R} \cdot \mathcal{F}[v]](\boldsymbol{x}), \tag{3.4}$$

where $\boldsymbol{R} \in \mathbb{C}^{n\times n}$ is the learnable frequency domain filter parameter matrix. In the actual calculation, we use the discrete Fourier transform (DFT), and introduce a frequency truncation mechanism.

$$(\mathcal{K}v)(\boldsymbol{x}) \approx \mathcal{F}^{-1}\left[\sum_{|\boldsymbol{k}_x|<k_{\max},|\boldsymbol{k}_y|<k_{\max}} \boldsymbol{R}_{\boldsymbol{k}_x,\boldsymbol{k}_y} \cdot \hat{v}_{\boldsymbol{k}_x,\boldsymbol{k}_y}\right](\boldsymbol{x}) \tag{3.5}$$

The choice of truncation frequency $k_{\max}$ is based on theoretical analysis and numerical experiments, we determined through spectral analysis that $k_{\max} = 16$.

3.1.2. Spectral convolution layer

In the FNO framework, spectral convolutional layers are the core computational units, which are mathematically expressed as follows: Given an input feature map $v \in \mathbb{R}^{N\times d_{in}}$, where N is the number of spatial discretions and d_{in} is the number of input channels, we define the transformation of spectral convolutional layers as follows:

$$\mathcal{S}(v) = \mathcal{F}^{-1}\left[\mathcal{P} \cdot \mathcal{F}[v]\right], \tag{3.6}$$

where $\mathcal{P} \in \mathbb{C}^{d_{out}\times d_{in}\times k_{\max}\times k_{\max}}$ is a complex-valued parameter tensor that performs a linear transformation on features in the frequency domain. In particular, for each pair of input channel i and output channel j, the parameter $\mathcal{P}_{j,i} \in \mathbb{C}^{k_{\max}\times k_{\max}}$ is nonzero, the high frequency part is zero, and high frequency calculating is truncated.

For a two-dimensional problem, given an input $v \in \mathbb{R}^{N_x\times N_y\times d_{in}}$, the forward propagation process of the spectral convolutional layer consists of the following steps;

1) Calculat a two-dimensional discrete Fourier transform for each channel i: $\hat{v}_i = \mathcal{F}[v_i] \in \mathbb{C}^{N_x\times N_y}$.

2) Perform a complex-valued linear transformation in the frequency domain: for each output channel $j \in \{1, 2, \ldots, d_{out}\}$,

$$\hat{w}_j = \sum_{i=1}^{d_{in}} \hat{v}_i \odot \mathcal{P}_{j,i}, \tag{3.7}$$

where $\odot$ indicates that the element-level complex multiplication: $(a + bi) \odot (c + di) = (ac - bd) + (ad + bc)i$£¬and $\mathcal{P}_{j,i}$ in $[-k_{\max}, k_{\max}]^2$ area outside the matrix is zero.

3) Perform an inverse Fourier transform for each output channel j : $w_j = \mathcal{F}^{-1}[\hat{w}_j] \in \mathbb{R}^{N_x\times N_y}$.

Because the parameterization is performed only in the low-frequency region, the number of parameters of the spectral convolution laystructure design is $O(d_{in} \times d_{out} \times k_{\max}^2)$, and the number of parameters of the spectral convolution layer is $O(d_{in} \times d_{Max}^2)$. $O(d_{in} \times d_{out} \times k_{\max}^2)$ is much smaller than $O(d_{in} \times d_{out} \times N_x \times N_y)$ of the spatial domain convolution, while the computational complexity is reduced to $O(N_x N_y \log(N_x N_y))$.

The flow of the spectral convolutional layer is shown in Figure 4.

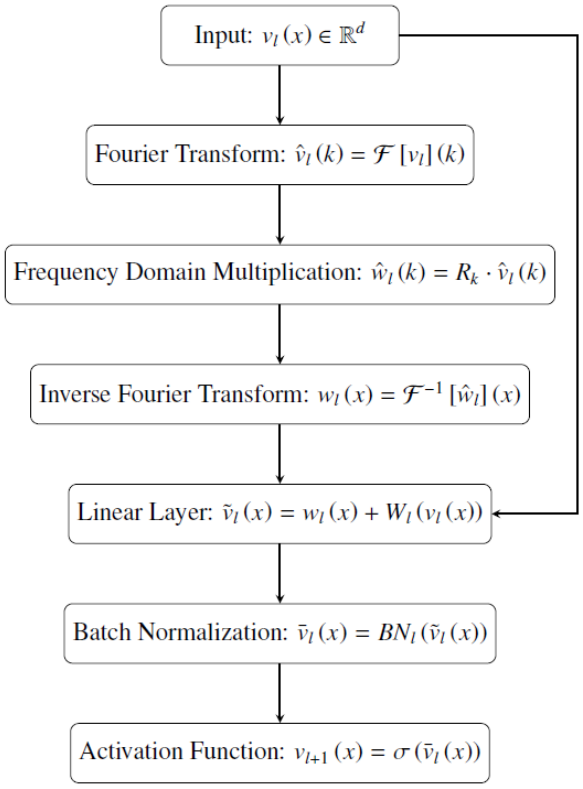


Figure 4. Detailed flow of spectral convolution operation.

3.1.3. Enhanced FNO network architecture

The enhanced FNO two-dimensional network we designed can be mathematically expressed as a series of function composition operations which are composed of the following modules.

1) Projection layer: Maps the input coordinates $(\boldsymbol{x}, t) \in \mathbb{R}^{d+1}$ to a high-dimensional feature space,

$$\boldsymbol{v}_0 = \sigma(\boldsymbol{W}_0(\boldsymbol{x}, t) + \boldsymbol{b}_0) \in \mathbb{R}^{N \times d_0}, \tag{3.8}$$

where $\boldsymbol{W}_0 \in \mathbb{R}^{d_0 \times (d+1)}$£¬$\boldsymbol{b}_0 \in \mathbb{R}^{d_0}$, σ is a nonlinear activation function.

2) Sequence of spectral convolution blocks: Cascaded L spectral convolution blocks, each containing a combination of spectral convolution, linear layer, batch normalization, and nonlinear activation expressed as

$$\boldsymbol{v}_{l+1} = \sigma\left(\mathrm{BN}_l\left(\mathcal{S}_l(\boldsymbol{v}_l) + \boldsymbol{W}_l \boldsymbol{v}_l\right)\right), \quad l = 0, 1, 2, \ldots, L-1, \tag{3.9}$$

where $\mathcal{S}_l$ is the l th spectral convolution layer, $\boldsymbol{W}_l$ is the 1×1 point convolution implemented by the parameter matrix, and $\mathrm{BN}_{\ l}$ is the batch normalization operation. This structure design combines the ability of spectral convolution to capture global features and the advantage of point convolution to learn local features.

3) Output mapping network: Converts feature maps into solution function space.

$$\boldsymbol{h}_1 = \sigma(\boldsymbol{W}_L \boldsymbol{v}_L + \boldsymbol{b}_L), \tag{3.10}$$

$$\boldsymbol{h}_2 = \sigma(\boldsymbol{W}_{L+1}\boldsymbol{h}_1 + \boldsymbol{b}_{L+1}), \tag{3.11}$$

$$(u_1, u_2, \ldots, u_m) = \boldsymbol{W}_{L+2}\boldsymbol{h}_2 + \boldsymbol{b}_{L+2}. \tag{3.12}$$

Network hyperparameters are selected based on error analysis and computational efficiency considerations.

Frequency pattern truncation: $k_{\max,1} = k_{\max,2} = 16$, determined from the spectral analysis of the solution. Network width: $d_0 = d_1 = \ldots = d_{L-1} = 32$, which balances expressive power and computational cost. Dimension of fully connected layer: $\boldsymbol{W}_L : 32 \to 128$, $\boldsymbol{W}_{L+1} : 128 \to 64$, $\boldsymbol{W}_{L+2} : 64 \to m$, where m is the number of output components.

3.1.4. Batch normalization with GELU activation

For the systems with different time scales, batch normalization plays a key role in improving training stability and accelerating convergence. Its mathematical definition is as follows:

$$\mathrm{BN}(\boldsymbol{x}) = \boldsymbol{\gamma} \odot \frac{\boldsymbol{x} - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \epsilon}} + \boldsymbol{\beta}, \tag{3.13}$$

where $\boldsymbol{\mu}_B$ and $\boldsymbol{\sigma}_B^2$ are the mean and variance within the mini-batch, respectively. $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learnable scaling and offset parameters, and ϵ is a small constant that prevents division by zero.

For the activation function, we use the Gaussian-error linear unit (GELU), whose mathematical expression is as follows:

$$\mathrm{GELU}(x) = x \cdot \Phi(x) = x \cdot \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right), \tag{3.14}$$

where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution, and erf is the error function:

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt. \tag{3.15}$$

We choose GELU for two reasons. First, the rectified linear unit (GELU) is smooth (infinitely differentiable). This is important for frequency-domain learning. Non-smooth functions like (the rectified linear unit) ReLU create high-frequency components. These do not work well with our frequency truncation. Second, Tanh has vanishing gradient problems for large inputs, while GELU keeps stable gradients across all layers. This choice is also consistent with the original FNO paper [20].

3.2. The construction of a fourth-order finite difference scheme

In FNO-PINN, we need accurate partial derivatives to compute PDE residuals. Standard PINNs use automatic differentiation (AD), but we use a high-order finite difference (FD) scheme, instead. We choose FD for two reasons. First, the fourth-order compact scheme gives $O(h^4)$ accuracy for spatial derivatives. This is better than point-wise AD on discrete grids. Also, using FD-based residuals ensures consistency with the discrete reference solutions used for data supervision, thereby avoiding potential discretization mismatch. The effectiveness of this FD-based approach over AD is validated by ablation studies in Section 4.

3.2.1. Interior point method

Taking the following simple system of equations as an example, a high-order finite difference scheme is proposed.

$$\begin{cases} \dfrac{\partial u}{\partial t} = c\dfrac{\partial^2 u}{\partial x^2} + f(x,t), & \Omega = [a,b]\times[0,T], \\ u(x,0) = g(x), & a \le x \le b, \\ u(a,t) = g_1(t), \quad u(b,t) = g_2(t), & 0 \le t \le T, \end{cases} \tag{3.16}$$

where the functions $g_1(t)$, $g_2(t)$, $g(x)$, and $f(x,t)$ are known and sufficiently smooth. To construct a finite difference approximation for the differential problem Eq (3.16), the spatial domain $[a,b]$ is uniformly discretized by selecting a positive integer N and setting $h = (b-a)/N$. Then the temporal domain $[0,T]$ is uniformly discretized by choosing a positive integer S and setting $\tau = T/S$. Let f_i^n denote the value of the unknown function at the node (x_i, t_n) with $x_i = a + (i-1)h, i = 1, \cdots, N+1$, $x_1 = a, x_{N+1} = b,\ t_n = n\tau, n = 0, 1, \cdots, S$.

For derivation, we define the second-order central difference operator

$$\delta_x^2 u_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}, \tag{3.17}$$

$$\delta_x u_i = \frac{u_{i+1} - u_{i-1}}{2h}. \tag{3.18}$$

The diffusion term is discretized using a fourth-order compact difference scheme for the interior points

$$\frac{\partial^2 u}{\partial x^2} = \frac{\delta_x^2 u}{1 + \frac{h^2}{12}\delta_x^2} + O(h^4). \tag{3.19}$$

Substituting Eq (3.19) into Eq (3.16) yields

$$\frac{\partial u_i}{\partial t} = c\frac{\delta_x^2}{1 + \frac{h^2}{12}\delta_x^2}u_i + f_i + O(h^4); \tag{3.20}$$

after rearrangement, we obtain

$$\frac{\partial u_i}{\partial t} + \frac{h^2}{12}\delta_x^2\frac{\partial u_i}{\partial t} = c\delta_x^2 u_i + f_i + \frac{h^2}{12}\delta_x^2 f_i + O(h^4). \tag{3.21}$$

Substituting Eq (3.17) into Eq (3.21) and rearranging, we arrive at

$$\frac{1}{12}\left(\frac{\partial u_{i+1}}{\partial t} + 10\frac{\partial u_i}{\partial t} + \frac{\partial u_{i-1}}{\partial t}\right) = \frac{c}{h^2}(u_{i+1} - 2u_i + u_{i-1}) + \frac{1}{12}(f_{i+1} + 10f_i + f_{i-1}) + O(h^4). \tag{3.22}$$

3.2.2. Boundary format

A fourth-order boundary condition is applied to the spatial second derivative at the boundary points [24]. The fourth-order scheme for the left boundary of the second derivative is given by

$$\frac{14}{12}u_1'' - \frac{5}{12}u_2'' + \frac{4}{12}u_3'' - \frac{1}{12}u_4'' = \frac{1}{h^2}\big(u_1 - 2u_2 + u_3\big). \tag{3.23}$$

The fourth-order scheme for the right boundary of the second derivative is given by

$$\frac{14}{12}u_{N+1}'' - \frac{5}{12}u_N'' + \frac{4}{12}u_{N-1}'' - \frac{1}{12}u_{N-2}'' = \frac{1}{h^2}\big(u_{N+1} - 2u_N + u_{N-1}\big). \tag{3.24}$$

Expressed in matrix form, it is $B_1U'' = \frac{1}{h^2}B_2U$, where

$$B_1 = \begin{pmatrix} \frac{14}{12} & -\frac{5}{12} & \frac{4}{12} & -\frac{1}{12} & \cdots & 0 & 0 & 0 & 0 \\ \frac{1}{12} & \frac{5}{6} & \frac{1}{12} & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{12} & \frac{5}{6} & \frac{1}{12} & \cdots & 0 & 0 & 0 & 0 \\ & & & & \ddots & & & & \\ 0 & 0 & 0 & 0 & \cdots & \frac{1}{12} & \frac{5}{6} & \frac{1}{12} & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & \frac{1}{12} & \frac{5}{6} & \frac{1}{12} \\ 0 & 0 & 0 & 0 & \cdots & -\frac{1}{12} & \frac{4}{12} & -\frac{5}{12} & \frac{14}{12} \end{pmatrix}_{(N+1)\times(N+1)} \quad \mathrm{U}'' = \begin{pmatrix} u_1'' \\ u_2'' \\ u_3'' \\ \vdots \\ u_N'' - 1 \\ u_N'' \\ u_N'' \end{pmatrix}_{N+1}$$

$$B_2 = \begin{pmatrix} -2 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 1 & -2 & 1 & \cdots & 0 & 0 & 0 \\ 0 & 1 & -2 & \cdots & 0 & 0 & 0 \\ & & & \ddots & & & \\ 0 & 0 & 0 & \cdots & -2 & 1 & 0 \\ 0 & 0 & 0 & \cdots & 1 & -2 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -2 \end{pmatrix}_{(N+1)\times(N+1)}$$

Obviously $U'' = B_1^{-1}\left(\frac{1}{h^2}B_2U\right)$.

The semi-discrete equation for the fourth-order difference scheme of Eq (3.16) is obtained as

$$\frac{\mathbf{dU}}{dt} = vB_1^{-1}\frac{1}{h^2}B_2\mathbf{U} + \mathbf{f}(t), \tag{3.25}$$

where, $\mathbf{U} = (u_1, u_2, u_3, \ldots, u_{N+1})^T$, $\quad \mathbf{f}(t) = (f_1, f_2, f_3, \ldots, f_{N+1})^T$.

For later use in the FNO-PINN framework, we rewrite Eq (3.25) in an operator form. Define the discrete second-derivative operator

$$L_hU := B_1^{-1}\left(\frac{1}{h^2}B_2U\right), \tag{3.26}$$

so that the semi-discrete scheme reads

$$\frac{dU}{dt} = c\,L_hU + f(t).$$

In the FNO-PINN setting, let $t^n = n\Delta t$ $(n = 0, \ldots, N_t)$ be the temporal grid, and denote by $U_\theta^n = U_\theta(t^n) := (u_{\theta,1}^n, \ldots, u_{\theta,N+1}^n)^T$ the network prediction at time t^n. At a grid point (x_i, t^n), the discrete PDE residual is defined as

$$r_i^n(\theta) = \mathcal{D}_t[u_\theta](x_i, t^n) - c\,(L_hU_\theta^n)_i - f_i^n, \tag{3.27}$$

where $(L_hU_\theta^n)_i$ denotes the i-th component of $L_hU_\theta^n$, and $\mathcal{D}_t[u_\theta]$ represents the temporal derivative, which can be computed by two approaches:

Finite difference (FD) approach (default):

$$\mathcal{D}_t[u_\theta](x_i, t^n) \approx \frac{-u_{\theta,i}^{n+2} + 8u_{\theta,i}^{n+1} - 8u_{\theta,i}^{n-1} + u_{\theta,i}^{n-2}}{12\Delta t}. \tag{3.28}$$

Automatic differentiation (AD) approach (baseline):

$$\mathcal{D}_t[u_\theta](x_i, t^n) = \frac{\partial u_\theta}{\partial t}(x_i, t^n). \tag{3.29}$$

In our proposed FNO-PINN framework, we adopt the FD approach as the default method. The AD approach serves as a baseline for the ablation studies presented in Section 4.

The discrete PDE residual loss is then given by

$$L_{\text{PDEs}}(\theta) = \frac{1}{N_{\text{int}}}\sum_{(i,n)\in\mathcal{I}_{\text{int}}}\left(r_i^n(\theta)\right)^2, \tag{3.30}$$

where $\mathcal{I}_{\text{int}}$ denotes the index set of interior space time collocation points and $N_{\text{int}} = |\mathcal{I}_{\text{int}}|$.

Remark on the role of boundary formulas in FNO-PINN. The fourth-order boundary formulas in Eqs (3.23)–(3.25), together with the interior compact scheme, define a complete high-order finite difference discretization for Eq (3.16). In this work, they play two related but distinct roles:

Finite difference reference solver. When Eq (3.25) is used as a classical finite difference method to generate reference solutions, all grid nodes, including boundary and boundary-adjacent points, participate in the semi-discrete equation

$$\frac{dU}{dt} = c\,L_hU + f(t),$$

with the Dirichlet boundary conditions $u(a,t) = g_1(t)$ and $u(b,t) = g_2(t)$ imposed on $U(t)$.

In the FNO-PINN method, the compact fourth-order operator L_h is used to approximate the second spatial derivatives at interior grid points. The Dirichlet (or periodic) boundary conditions are not enforced by adding PDE residuals at the boundary. Instead, they are enforced by the boundary loss L_{BC}. In contrast, the full matrix-based scheme is used in the finite difference reference solver. In that case, all grid points follow the semi-discrete equation.

3.2.3. Discretization in the temporal direction

The temporal discretization employs a third-order total variation diminishing (TVD) Runge-Kutta method [25]. Let

$$\mathbf{b_1} = \frac{1}{h^2} B_2.$$

The third-order explicit Runge-Kutta scheme is obtained as follows:

$$\begin{cases} L_1 = u + td_1 \\ d_1 = Ru\,(u, B_1, \mathbf{b_1}) \\ L_2 = \frac{3}{4}u + \frac{1}{4}(L_1 + td_2) \\ d_2 = Ru\,(L_1, B_1, \mathbf{b_1}) \\ L_3 = \frac{1}{3}u + \frac{2}{3}(L_2 + td_3) \\ d_3 = Ru\,(L_2, B_1, \mathbf{b_1}) \end{cases}. \tag{3.31}$$

The above equation represents the discretized results of all terms at time step n.

3.3. Stability analysis of the scheme

In this subsection, we employ the Fourier analysis method to conduct a stability analysis of the linearized scheme. For this purpose, we assume that the boundary conditions of Eq (3.16) are perfectly satisfied that there are no errors in the right-side term $f(t)$. The error produced in computing $\varepsilon_i^n = u_i^n - u(x_i, t_n)$ denoted by u_i^n, $u(x_i, t_n)$ is the exact solution.

The Runge-Kutta method is obtained by

$$u_i^{n+1} = u_i^n + t\frac{\delta_x^2}{1 + \frac{h^2}{12}\delta_x^2} u_i^n + \frac{1}{2}t^2 \frac{\left(\delta_x^2\right)^2}{\left(1 + \frac{h^2}{12}\delta_x^2\right)^2} u_i^n + \frac{1}{6}t^3 \frac{\left(\delta_x^2\right)^3}{\left(1 + \frac{h^2}{12}\delta_x^2\right)^3} u_i^n. \tag{3.32}$$

Substituting ε_i^n into Eq (3.32) yields the following error equation:

$$\varepsilon_i^{n+1} = \varepsilon_i^n + t\frac{\delta_x^2}{1 + \frac{h^2}{12}\delta_x^2} \varepsilon_i^n + \frac{1}{2}t^2 \frac{\left(\delta_x^2\right)^2}{\left(1 + \frac{h^2}{12}\delta_x^2\right)^2} \varepsilon_i^n + \frac{1}{6}t^3 \frac{\left(\delta_x^2\right)^3}{\left(1 + \frac{h^2}{12}\delta_x^2\right)^3} \varepsilon_i^n.$$

Upon simplification, we obtain

$$\begin{aligned} &\varepsilon_i^{n+1} + \frac{h^2}{4}\delta_x^2\varepsilon_i^{n+1} + \frac{3h^4}{144}\delta_x^4\varepsilon_i^{n+1} + \frac{h^6}{1728}\delta_x^6\varepsilon_i^{n+1} = \varepsilon_i^n + \frac{h^2}{4}\delta_x^2\varepsilon_i^n + \frac{3h^4}{144}\delta_x^4\varepsilon_i^n + \frac{h^6}{1728}\delta_x^6\varepsilon_i^n + \\ &t\delta_x^2\varepsilon_i^n + \frac{th^2}{6}\delta_x^4\varepsilon_i^n + \frac{th^4}{144}\delta_x^6\varepsilon_i^n + \frac{1}{2}t^2\delta_x^4\varepsilon_i^n + \frac{1}{12}t^2h^2\delta_x^6\varepsilon_i^n + \frac{1}{288}t^2h^4\delta_x^8\varepsilon_i^n + \frac{1}{6}t^2\delta_x^6\varepsilon_i^n \end{aligned}, \tag{3.33}$$

where $\varepsilon_i^n = \psi^n e^{I\Gamma x_i}$. Denote A as equal to the left side in the Eq (3.33):

$$A = \psi^{n+1}(\frac{1924}{1728} + \frac{606}{1728}\cos\Gamma h + \frac{60}{1728}\cos 2\Gamma h + \frac{2}{1728}\cos 3\Gamma h), \tag{3.34}$$

denote B as equal to the right side in the Eq (3.33).

$$\begin{aligned} B =&\psi^{n+1}[\left(\frac{1924}{1728} - \frac{43}{38}\tau + \frac{29}{18}\tau^2 - \frac{10}{3}\frac{\tau^2}{h^2}\right) + \left(\frac{303}{1728} + \frac{16}{144}\tau + \frac{1168}{1728}\tau^2 + \frac{10}{3}\frac{\tau^2}{h^2}\right) \times 2\cos\Gamma h + \\ &\left(\frac{30}{1728} + \frac{18}{144}\tau + \frac{28}{288}\tau^2 - \frac{\tau^2}{h^2}\right) \times 2\cos 2\Gamma h + \left(\frac{1}{1728} + \frac{1}{144}\tau + \frac{16}{288}\tau^2 + \frac{1}{6}\frac{\tau^2}{h^2}\right) \times 2\cos 3\Gamma h + \\ &\frac{2}{288}\tau^2\cos 4\Gamma h] \end{aligned} \tag{3.35}$$

The growth factor is denoted by G shown as follows:

$$G = \frac{\psi^{n+1}}{\psi^n} = \frac{B}{A}. \tag{3.36}$$

For any mesh ratio $\gamma = \frac{\tau}{h^2}$, $|G| \leq 1$ is satisfied, hence the difference scheme is stable.

3.4. Loss functions of FNO-PINN

In this subsection, we describe the loss functions for the two training phases of FNO-PINN. We consider a general PDE of the form

$$\mathcal{N}[u](x,t) = 0, \qquad (x,t) \in \Omega := D_x \times D_t \tag{3.37}$$

with initial and boundary conditions

$$u(x,t_0) = u_0(x), \qquad x \in D_x, \qquad u(x,t) = g(x,t), \quad (x,t) \in \partial D_x \times D_t. \tag{3.38}$$

In our FNO-PINN framework, all spatial and temporal derivatives in $\mathcal{N}$ are evaluated by the high-order FD scheme introduced in Section 3.2, so that the discrete operator $\mathcal{N}_h$ is fully determined by the FD stencils. For comparison, we also test a standard AD method in Section 4. In this case, we compute derivatives using AD instead of FD. The loss functions stay the same.

3.4.1. Discrete space-time grid and notation

Let $\{x_j\}_{j=0}^{N_x-1} \subset D_x$ and $\{t_k\}_{k=0}^{N_t-1} \subset D_t$ denote a uniform space–time grid

$$x_j = x_{\min} + j\Delta x, \quad t_k = t_{\min} + k\Delta t,$$

with spatial and temporal steps Δx and Δt, respectively. We denote by $u_{j,k} = u_\theta(x_j, t_k)$ the prediction of a neural network u_θ at the grid point (x_j, t_k), where θ represents the model parameters. For later use, we introduce the following index sets:

$$\mathcal{I}_{\text{int}} \subset \{0,\dots,N_x-1\} \times \{0,\dots,N_t-1\} \quad \text{(interior points)},$$

$$\mathcal{I}_0 = \{0,\dots,N_x-1\} \quad \text{(initial line at } t = t_0),$$

$$\mathcal{I}_{\text{bc}} \subset \{0,\dots,N_x-1\} \times \{0,\dots,N_t-1\} \quad \text{(spatial boundary points)}.$$

3.4.2. Discrete PDE residual loss

At an interior grid point (x_j, t_k), the discrete PDE residual is defined as

$$r_{j,k} := \mathcal{N}_h[u_\theta](x_j, t_k), \tag{3.39}$$

where $\mathcal{N}_h$ denotes the FD-based discrete version of the differential operator $\mathcal{N}$ obtained from the fourth-order finite difference scheme (see Section 3.2). In other words, all derivatives in $\mathcal{N}$ are replaced by their FD approximations on the grid.

As a representative example, for the viscous Burgers equation

$$\partial_t u + u\,\partial_x u - \nu\,\partial_{xx} u = 0,$$

we have

$$r_{j,k} = \partial_t u_\theta(x_j, t_k) + u_\theta(x_j, t_k)\,\partial_x u_\theta(x_j, t_k) - \nu\,\partial_{xx} u_\theta(x_j, t_k), \tag{3.40}$$

where the derivatives are approximated by the fourth-order compact FD formulas, for example

$$\partial_t u_\theta(x_j, t_k) \approx \frac{-u_{j,k+2} + 8u_{j,k+1} - 8u_{j,k-1} + u_{j,k-2}}{12\Delta t}, \tag{3.41}$$

$$\partial_{xx} u_\theta(x_j, t_k) \approx \frac{-u_{j-2,k} + 16u_{j-1,k} - 30u_{j,k} + 16u_{j+1,k} - u_{j+2,k}}{12\Delta x^2}, \tag{3.42}$$

with boundary closures handled as detailed in Section 3.2.2. Similar formulas are used for the spatial derivatives in the coupled diffusion system and the nonlinear coupled system.

The PDE residual loss is defined as the mean squared residual over all interior grid points:

$$L_{\mathrm{PDE}}(\theta) = \frac{1}{|\mathcal{I}_{\mathrm{int}}|} \sum_{(j,k)\in\mathcal{I}_{\mathrm{int}}} r_{j,k}^2. \tag{3.43}$$

For multi-component systems $u = (u^{(1)}, \ldots, u^{(M)})^\top$, the residual is vector-valued as $r = (r^{(1)}, \ldots, r^{(M)})^\top$, and we set

$$L_{\mathrm{PDE}}(\theta) = \frac{1}{|\mathcal{I}_{\mathrm{int}}|} \sum_{(j,k)\in\mathcal{I}_{\mathrm{int}}} \sum_{m=1}^{M} (r_{j,k}^{(m)})^2.$$

3.4.3. Data loss

As discussed in Section 3.1, we may have access to a set of labeled space–time samples

$$\mathcal{D}_{\mathrm{data}} = \{(x_m, t_m, u^{\mathrm{data}}(x_m, t_m))\}_{m=1}^{N_{\mathrm{data}}},$$

where u^{data} comes from high-accuracy numerical solvers or analytical solutions. The data loss is defined as

$$L_{\mathrm{data}}(\theta) = \frac{1}{N_{\mathrm{data}}} \sum_{m=1}^{N_{\mathrm{data}}} \big(u_\theta(x_m, t_m) - u^{\mathrm{data}}(x_m, t_m)\big)^2. \tag{3.44}$$

In the second training phase, the output of the first-stage FNO can also be used as an "auxiliary data" source, and is incorporated through an anchor loss defined below.

3.4.4. Initial and boundary condition losses

On the initial time slice $t = t_0$, we enforce the initial condition $u(x, t_0) = u_0(x)$ via the discrete loss

$$L_{\mathrm{IC}}(\theta) = \frac{1}{|\mathcal{I}_0|} \sum_{j \in \mathcal{I}_0} \big(u_\theta(x_j, t_0) - u_0(x_j)\big)^2. \tag{3.45}$$

For Dirichlet-type boundary conditions $u(x, t) = g(x, t)$ on $\partial D_x \times D_t$, the boundary loss reads

$$L_{\mathrm{BC}}(\theta) = \frac{1}{|\mathcal{I}_{\mathrm{bc}}|} \sum_{(j,k) \in \mathcal{I}_{\mathrm{bc}}} \big(u_\theta(x_j, t_k) - g(x_j, t_k)\big)^2. \tag{3.46}$$

Periodic boundary conditions mean that both the function and its derivatives are periodic.

$$\mathcal{L}_{BC}^P(\theta) = \sum_{i=1}^{m} \frac{1}{|\partial\Omega_T|} \int_{\partial\Omega_T} \left[(u_i^\theta(x_{\min}, t) - u_i^\theta(x_{\max}, t))^2 + \left(\frac{\partial u_i^\theta}{\partial x}(x_{\min}, t) - \frac{\partial u_i^\theta}{\partial x}(x_{\max}, t) \right)^2 \right] ds\, dt. \tag{3.47}$$

In the actual implementation, we compute the boundary loss discretized. For example, for periodic boundary conditions, the discrete form is as follows:

$$\begin{aligned} \mathcal{L}_{BC}^P(\theta) \approx \sum_{i=1}^{m} \frac{1}{N_t} \sum_{k=0}^{N_t-1} \Bigg[& (u_i^\theta(x_0, t_k) - u_i^\theta(x_{N_x-1}, t_k))^2 \\ & + \left(\frac{u_i^\theta(x_1, t_k) - u_i^\theta(x_0, t_k)}{\Delta x} - \frac{u_i^\theta(x_{N_x-1}, t_k) - u_i^\theta(x_{N_x-2}, t_k)}{\Delta x} \right)^2 \Bigg] \Delta t. \end{aligned} \tag{3.48}$$

3.4.5. Anchor loss for the second phase

Let u_{θ^*} denote the operator learned by the first-stage FNO model. In the second phase, to prevent the PINN from deviating too far from this learned operator while still enforcing the PDE and physical constraints, we introduce an anchor loss on a set of anchor points $\{(x_m, t_m)\}_{m=1}^{N_{\mathrm{anc}}}$:

$$L_{\mathrm{anchor}}(\phi) = \frac{1}{N_{\mathrm{anc}}} \sum_{m=1}^{N_{\mathrm{anc}}} \big(u_\phi(x_m, t_m) - u_{\theta^*}(x_m, t_m)\big)^2, \tag{3.49}$$

where u_ϕ is the PINN used in the second phase.

3.4.6. Loss composition in the two training phases

We now explicitly state the loss functions used in the two stages of FNO-PINN.

Phase 1: operator learning with FNO. In the first phase, we train an FNO-based operator u_θ by minimizing

$$J_{\mathrm{op}}(\theta) = \lambda_{\mathrm{PDE}}^{(1)} L_{\mathrm{PDE}}^{(1)}(\theta) + \lambda_{\mathrm{data}}^{(1)} L_{\mathrm{data}}^{(1)}(\theta) + \lambda_{\mathrm{IC}}^{(1)} L_{\mathrm{IC}}^{(1)}(\theta) + \lambda_{\mathrm{BC}}^{(1)} L_{\mathrm{BC}}^{(1)}(\theta), \tag{3.50}$$

where the superscript (1) indicates that the losses are evaluated using the first-stage FNO model. The coefficients $\lambda_{\mathrm{PDE}}^{(1)}, \lambda_{\mathrm{data}}^{(1)}, \lambda_{\mathrm{IC}}^{(1)}, \lambda_{\mathrm{BC}}^{(1)}$ are constant weights. Depending on the availability of high-fidelity data, some terms may be deactivated in certain examples (e.g, setting $\lambda_{\mathrm{PDE}}^{(1)} = 0$ when only data supervision is used). The specific choices for each numerical example are detailed in Section 4.

Phase 2: fine-tuning with PINNs. In the second phase, we use a PINN u_ϕ initialized from or guided by the first-stage model and minimize

$$J_{\text{fine}}(\phi) = \lambda^{(2)}_{\text{PDE}} L^{(2)}_{\text{PDE}}(\phi) + \lambda^{(2)}_{\text{IC}} L^{(2)}_{\text{IC}}(\phi) + \lambda^{(2)}_{\text{BC}} L^{(2)}_{\text{BC}}(\phi) + \lambda^{(2)}_{\text{data}} L^{(2)}_{\text{data}}(\phi) + \lambda_{\text{anc}} L_{\text{anchor}}(\phi), \tag{3.51}$$

where the superscript (2) indicates that the losses are evaluated using the second-stage PINN. In this phase, the PDE residuals are also evaluated by the FD scheme discussed in Section 3.2, and the dynamic weight adjustment strategy introduced in Section 3.4.5 acts on the coefficients $\lambda^{(2)}_{\text{PDE}}$, $\lambda^{(2)}_{\text{IC}}$, and $\lambda^{(2)}_{\text{BC}}$ to alleviate imbalance among different loss terms, especially for multi-component systems with different time scales. The concrete weight schedules and parameter values used in each example are summarized in Section 4.

Remark. For comparison, we also implement a baseline version where the derivatives in $\mathcal{N}$ are computed by AD instead of FD while keeping the same loss definitions (3.43)–(3.51). The corresponding results are reported in Section 4 to highlight the advantages of FD-based FNO-PINN.

3.5. Finite-difference-based computation of derivatives

As stated in Section 3.4, the core of our FNO-PINN framework is to evaluate the PDE operator N by a high-order FD scheme on a discrete space–time grid. In this subsection, we summarize in a problem-independent manner how the derivatives of the neural operator are computed by FD and how this is integrated into both training phases.

Let $u_\theta(x, t)$ denote the neural operator, with θ the trainable parameters. In the first phase, u_θ represents the enhanced FNO model (Section 3.1); in the second phase, u_θ is the PINN used for fine-tuning (Section 3.6.4). On the uniform grid $\{(x_j, t_k)\}$ introduced in Section 3.4.1, we write

$$u_{j,k} = u_\theta(x_j, t_k), \qquad (x_j, t_k) \in D_x \times D_t,$$

FD-based discrete operator N_h. Consider a general PDE of the form

$$N[u](x, t) = 0,$$

where N involves spatial and temporal derivatives of u (and possibly of multiple components in the coupled cases). On the discrete grid, we construct a finite-difference approximation N_h of N by replacing all derivatives by the high-order FD stencils introduced in Section 3.2.

More precisely, for each interior grid point $(x_j, t_k) \in \mathcal{I}_{\text{int}}$, we define the discrete residual

$$r_{j,k} = N_h[u_\theta](x_j, t_k), \tag{3.52}$$

where N_h is obtained from N by substituting the continuous derivatives, such as

$$\partial_t u, \quad \partial_{x_i} u, \quad \partial_{x_i x_\ell} u, \quad \ldots,$$

with the corresponding fourth-order FD approximations on the grid (and using the boundary closures described in Section 3.2.2 near the spatial and temporal boundaries). For multi-component systems $u = (u^{(1)}, \ldots, u^{(M)})^\top$, the operator N_h acts component-wise and yields a vector-valued residual $r_{j,k} = (r^{(1)}_{j,k}, \ldots, r^{(M)}_{j,k})^\top$.

The PDE loss L_{PDE} used in both training phases is then computed from these discrete residuals according to the fully discrete formulation in Section 3.4.2.

$$L_{\mathrm{PDE}}(\theta) = \frac{1}{|\mathcal{I}_{\mathrm{int}}|} \sum_{(j,k)\in\mathcal{I}_{\mathrm{int}}} \left\| r_{j,k} \right\|_2^2,$$

with the Euclidean norm taken over components when $M > 1$. All other loss terms $L_{\mathrm{IC}}, L_{\mathrm{BC}}, L_{\mathrm{data}}, L_{\mathrm{anchor}}$ are defined in Section 3.4 and evaluated directly on the corresponding sets of grid points or sample points, without any additional discretization.

An important feature of our design is that the same FD-based operator N_h is used consistently in both stages of FNO-PINN: In the operator-learning phase (Phase 1), the FNO model u_θ is trained by minimizing $J_{\mathrm{op}}(\theta)$ in Eq (3.50), where L_{PDE} is built from the discrete residuals $r_{j,k} = N_h[u_\theta](x_j, t_k)$ as above, possibly combined with data loss and IC/BC losses depending on the example. In the fine-tuning phase (Phase 2), the PINN u_θ is trained by minimizing $J_{\mathrm{fine}}(\theta)$ in Eq (3.51). The PDE term in J_{fine} is evaluated by exactly the same FD-based operator N_h, now applied to the PINN output on the same type of space-time grid. The dynamic weight adjustment strategy acts on the coefficients of these loss terms, but does not change the definition of N_h.

Thus, from the viewpoint of derivative computation, both phases share a unified, high-order FD discretization of the PDE operator, and differ only in the underlying network architecture and loss weighting.

To assess the impact of the FD-based residuals, we additionally implement an automatic-differentiation (AD) baseline. In this variant, the overall training pipeline, including the two phases; the loss definitions $L_{\mathrm{PDE}}, L_{\mathrm{IC}}, L_{\mathrm{BC}}, L_{\mathrm{data}}, L_{\mathrm{anchor}}$; the dynamic weight strategy; and the optimization hyperparameters are kept identical to the FD-based FNO-PINN; only the internal evaluation of N is changed.

More precisely, in the AD-based baseline, we set

$$r_{j,k} = N[u_\theta](x_j, t_k),$$

where all derivatives in N are computed by automatic differentiation with respect to (x, t) at the grid points. The PDE loss still has the same discrete form

$$L_{\mathrm{PDE}}(\theta) = \frac{1}{|\mathcal{I}_{\mathrm{int}}|} \sum_{(j,k)\in\mathcal{I}_{\mathrm{int}}} \left\| r_{j,k} \right\|_2^2.$$

In this case, the discrete operator N_h coincides with the continuous operator N evaluated via AD on the grid. No additional loss terms or training stages are introduced; the baseline differs from our proposed method only in the way derivatives are obtained inside the same loss formulations.

Remark. Our proposed FNO-PINN method systematically uses the FD-based operator N_h in both the operator-learning and fine-tuning phases; the AD-based implementation is used solely as a baseline variant for ablation studies without changing the structure of the loss functions or the two-phase training strategy.

The comparison between FD-based and AD-based derivative computation is presented in the ablation studies of Section 4, which demonstrate the substantial accuracy advantage of the proposed FD-based approach.

3.6. Training strategies and post-processing techniques

In this subsection, we describe the optimization strategy, the two-stage training procedure of FNO-PINN, and the post-processing techniques used at the inference stage. Throughout this section, the loss components $L_{\text{PDE}}, L_{\text{IC}}, L_{\text{BC}}, L_{\text{data}}$, and L_{anchor} are those defined in Section 3.4, and all spatial/temporal derivatives in L_{PDE} are evaluated by the FD-based discrete operator N_h introduced in Section 3.2 and summarized in Section 3.5.

3.6.1. Optimization algorithms with learning-rate scheduling

We train the FNO-PINN models using the Adam optimizer combined with a cosine-annealing learning-rate schedule. The learning rate at epoch e is given by

$$\eta(e) = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\Big[1 + \cos\Big(\frac{e\pi}{E}\Big)\Big], \tag{3.53}$$

where $\eta_{\max} = 5 \times 10^{-4}$ and $\eta_{\min} = 10^{-6}$ are the maximum and minimum learning rates, respectively; e is the current epoch index; and E is the total number of epochs. This schedule works as follows: early in training, a large learning rate helps explore the parameter space quickly. Later, a small learning rate adjusts parameters finely. This improves convergence.

To prevent gradient explosion, we apply gradient clipping:

$$\nabla_\theta^{\text{clip}} J(\theta) = \nabla_\theta J(\theta)\ \min\Big(1,\ \frac{\gamma}{\|\nabla_\theta J(\theta)\|}\Big), \tag{3.54}$$

where γ is an upper bound on the gradient norm and is set to 1.0 in all experiments. We also use L^2 regularization to control model complexity and prevent overfitting:

$$J_{\text{reg}}(\theta) = J(\theta) + \lambda_{\text{reg}}\|\theta\|_2^2, \tag{3.55}$$

where $\lambda_{\text{reg}} = 10^{-6}$ is the regularization strength. The same optimization settings (Adam, cosine annealing, gradient clipping, and L^2 regularization) are used in both training phases.

3.6.2. Two-stage training strategy of FNO-PINN

We now summarize the overall training pipeline of the proposed FNO-PINN framework. The key idea is to split the learning task into two stages: an operator-learning phase, where an enhanced FNO learns a global solution operator for a parameterized family of PDEs, and an instance-specific fine-tuning phase, where this operator is refined for a particular PDE instance with given initial and boundary conditions.

Both stages use the same FD-based operator $\mathcal{N}_h$ (Section 3.5) to compute the PDE residual loss L_{PDE}. The definitions of $L_{\text{IC}}, L_{\text{BC}}, L_{\text{data}}$, and L_{anchor} are the same as in Section 3.4. The two stages differ in their objectives and data usage, but not in the underlying discretization or basic loss components.

The FNO-PINN method uses a two-stage training strategy. Stage 1 learns the solution operator from multiple PDE instances. Stage 2 fine-tunes the model for a specific PDE problem. Table 1 summarizes this procedure.

Table 1. FNO-PINN two-stage training strategy.

FNO-PINN two-stage training strategy
Stage 1: Operator learning phase
1) Collect training data from a family of PDEs with different parameters.
2) Generate reference solutions using the finite-difference solver.
3) Initialize the FNO model parameters θ.
4) Train the model by minimizing the operator loss:
Loss Function: $J_{\text{op}}(\theta) = L_{\text{data}}(\theta) + \lambda_{\text{PDE}} L_{\text{PDE}}(\theta) + \lambda_{\text{IC}} L_{\text{IC}}(\theta) + \lambda_{\text{BC}} L_{\text{BC}}(\theta)$
5) Save the trained parameters as θ_0.
Stage 2: Instance-specific fine-tuning phase
1) Fix a specific PDE instance (coefficients, initial and boundary conditions).
2) Initialize with Stage 1 parameters: $\theta \leftarrow \theta_0$.
3. Fine-tune the model by minimizing the fine-tuning loss:
Loss Function: $J_{\text{fine}}(\theta) = L_{\text{PDE}}(\theta) + \lambda_{\text{IC}} L_{\text{IC}}(\theta) + \lambda_{\text{BC}} L_{\text{BC}}(\theta) + \lambda_{\text{anchor}} L_{\text{anchor}}(\theta)$
4) Output the fine-tuned solution u_θ.
5) (Optional) Apply post-processing to get the final solution $\widehat{u_\theta}$.

Stage 1 learns a general operator from many PDE instances. Stage 2 refines this operator for one specific problem. Both stages use the same finite-difference method to compute PDE residuals. The main difference is that Stage 2 includes an anchor loss L_{anchor} to prevent the solution from deviating too far from the Stage 1 result.

3.6.3. Operator learning phase

In the first stage, we train an enhanced FNO to learn the global solution operator of a parameterized family of PDEs. Let a denote the input coefficient or forcing field (and, if needed, other problem parameters), and let $u_\theta = G_\theta(a)$ be the corresponding FNO output. The training objective in this phase is

$$J_{\text{op}}(\theta) = L^{(1)}_{\text{data}}(\theta) + \lambda^{(1)}_{\text{PDE}} L^{(1)}_{\text{PDE}}(\theta) + \lambda^{(1)}_{\text{IC}} L^{(1)}_{\text{IC}}(\theta) + \lambda^{(1)}_{\text{BC}} L^{(1)}_{\text{BC}}(\theta), \tag{3.56}$$

where the superscript (1) indicates that the losses are evaluated using the first-stage FNO model.

$L^{(1)}_{\text{data}}$ measures the misfit between u_θ and available supervision data, typically the high-accuracy FD solutions on the same space-time grid. In the numerical experiments, for each PDE example, we use the FD solver described in Section 3.2 to generate a reference solution on the grid of $N_x = 128$ spatial points and $N_t = 64$ temporal points and define $L^{(1)}_{\text{data}}$ as in Eq (3.44). $L^{(1)}_{\text{PDE}}$ is the PDE residual loss defined in Section 3.4.2, constructed from the FD-based discrete operator $N_h[u_\theta]$ on the interior grid points. $L^{(1)}_{\text{IC}}$ and $L^{(1)}_{\text{BC}}$ are the initial- and boundary-condition losses defined in Section 3.4.4, evaluated directly on the corresponding subsets of grid points.

The weights $\lambda^{(1)}_{\text{PDE}}, \lambda^{(1)}_{\text{IC}}, \lambda^{(1)}_{\text{BC}}$ control the relative importance of the different physical constraints. In all examples, the same grid resolution $(N_x, N_t) = (128, 64)$ is used in Stage 1 as in Stage 2 for a fair

comparison with PINNs, and the specific values of these weights are reported in Section 4 for each PDE example.

In this operator-learning phase, the physics-based terms $L_{\text{PDE}}^{(1)}, L_{\text{IC}}^{(1)}$, and $L_{\text{BC}}^{(1)}$ guide the network to approximate a family of solution operators that satisfy the governing equations, while the data loss $L_{\text{data}}^{(1)}$ improves training efficiency and accuracy when high-fidelity reference solutions are available. This physics-informed, operator-level pre-training is particularly useful in scenarios where only limited observation data are available for each individual instance.

In the operator-learning phase, we generate training data using the fourth-order compact finite difference scheme (Section 3.2) combined with the third-order TVD Runge-Kutta time integrator. Unlike conventional operator learning methods that learn a parameter-to-solution mapping by sampling PDE parameters from a distribution, our FNO-PINN framework learns a coordinate-to-solution mapping $(x, t) \mapsto u(x, t)$ for a single PDE instance with fixed parameters. The training data consists of input space-time coordinates on a uniform grid ($N_x = 128$ spatial points $\times$ $N_t = 64$ temporal points = 8192 training samples) paired with reference solution values computed by the finite difference solver. This approach requires significantly less data (only one FD simulation per problem) while achieving high accuracy by combining the FNO's frequency-domain learning with the PINN's physics constraints.

3.6.4. Instance-specific fine-tuning phase

The second stage starts from the operator learned in the first stage and fine-tunes it for a given PDE instance with specific initial and boundary conditions. Let θ_0 denote the parameters of the trained FNO from the operator-learning phase, and initialize the fine-tuning network with $\theta = \theta_0$. For a fixed instance characterized by input a (e.g., fixed coefficients and specific initial/boundary data), the fine-tuned prediction is still denoted by $u_\theta = G_\theta(a)$.

The loss function in this phase is

$$J_{\text{fine}}(\theta) = L_{\text{PDE}}^{(2)}(\theta) + \lambda_{\text{IC}}^{(2)} L_{\text{IC}}^{(2)}(\theta) + \lambda_{\text{BC}}^{(2)} L_{\text{BC}}^{(2)}(\theta) + \lambda_{\text{anchor}} L_{\text{anchor}}(\theta) + \lambda_{\text{data}}^{(2)} L_{\text{data}}^{(2)}(\theta), \tag{3.57}$$

where the superscript (2) indicates that the losses are evaluated using the second-stage fine-tuned network.

Here $L_{\text{PDE}}^{(2)}, L_{\text{IC}}^{(2)}$, and $L_{\text{BC}}^{(2)}$ have the same definitions as in Stage 1, but are now evaluated for the specific instance. The additional term L_{anchor} is an anchor loss that regularizes the fine-tuning process around the pre-trained operator and is defined by

$$L_{\text{anchor}}(\theta) = \frac{1}{N_{\text{anc}}} \sum_{m=1}^{N_{\text{anc}}} \left\| G_\theta(a_m) - G_{\theta_0}(a_m) \right\|_2^2, \tag{3.58}$$

where $\{a_m\}_{m=1}^{N_{\text{anc}}}$ is a set of anchor inputs (e.g., sampled parameter settings or space–time locations for the same instance), and N_{anc} is the number of anchor points. The weight λ_{anchor} controls how strongly the fine-tuned model is constrained to stay close to the initial operator G_{θ_0}. In the numerical experiments, we set $N_{\text{anc}} = N_x N_t$ by using the full grid and report the specific values of λ_{anchor} and $\lambda_{\text{data}}^{(2)}$ for each example in Section 4.

This two-stage strategy works well for problems with multi-scale dynamics or different parameter ranges. Stage 1 learns the global behavior and the mapping from input a to solutions. Stage 2 adjusts

the model for a specific PDE with its exact initial and boundary conditions. In all experiments, the same FD-based operator N_h and loss definitions are used in both stages, and the difference lies in whether we learn a generic operator family or fine-tune for a single instance.

Remark. The FNO-PINN method connects classical operator-learning and standard PINN. If we skip Stage 2, the method becomes a physics-guided FNO. It learns a global solution operator using the FD-based residual L_{PDE} and optional data. If we skip Stage 1 and use random initialization in Stage 2, we get a standard PINN with the same FD-based discretization. The two-stage design thus combines the strengths of both paradigms: global operator generalization from Stage 1 and instance-specific accuracy from Stage 2.

3.6.5. Post-processing of neural-operator solutions

Finally, we explain the role of post-processing in FNO-PINN. Throughout both training phases, all loss terms defined in Section 3.4, namely $L_{\text{PDE}}, L_{\text{IC}}, L_{\text{BC}}, L_{\text{data}}$, and L_{anchor}, are evaluated directly on the raw network output $u_\theta(x,t)$ (FNO in Stage 1 and the fine-tuned network in Stage 2). We do not apply post-processing inside the loss functions or during gradient computation. Post-processing is only used at inference, after training ends. It enforces physical constraints in a simple way.

Let u_θ denote the trained network solution after the fine-tuning phase. We denote by $\widetilde{u}_\theta$ and $\widehat{u}_\theta$ the intermediate and final post-processed fields, respectively.

For problems with Dirichlet boundary conditions $u(x,t) = g(x,t)$ on $\partial\Omega$, we enforce the exact boundary values by the pointwise projection

$$\widetilde{u}_\theta(x,t) = \begin{cases} g(x,t), & x \in \partial\Omega, \\ u_\theta(x,t), & x \in \Omega \setminus \partial\Omega. \end{cases} \tag{3.59}$$

Similarly, for initial conditions $u(x,0) = h(x)$, we may set

$$\widetilde{u}_\theta(x,0) = h(x), \qquad x \in \Omega, \tag{3.60}$$

while leaving the interior points unchanged. In practice, L_{IC} and L_{BC} already make the network satisfy these conditions approximately, so the corrections in Eqs (3.59) and (3.60) are usually small.

For problems with periodic boundary conditions, the enhanced FNO architecture naturally handles periodic structure. To remove small non-periodic errors, we can rebuild a periodic approximation. We use a truncated Fourier expansion of the solution $\widetilde{u}_\theta(\cdot,t)$,

$$\widehat{u}_\theta(x,t) = \sum_{k\in\mathcal{K}} c_k(t)\,\varphi_k(x), \tag{3.61}$$

where $\{\varphi_k\}_{k\in\mathcal{K}}$ is a set of periodic basis functions (e.g., complex exponentials or sines and cosines) and $c_k(t)$ are the corresponding coefficients computed on the spatial grid.

Some PDEs preserve a global quantity, such as a spatial integral $\int_\Omega u(\cdot,t)\,\mathrm{d}x$ or a weighted sum of components. We can enforce this conservation at post-processing using global normalization. For instance, for a single scalar field u with conserved spatial average, one may set

$$\widehat{u}_\theta(x,t) = \widetilde{u}_\theta(x,t)\,\frac{\displaystyle\int_\Omega h(x)\,\mathrm{d}x}{\displaystyle\int_\Omega \widetilde{u}_\theta(x,t)\,\mathrm{d}x} \tag{3.62}$$

so that the integral of $\widehat{u_\theta}(\cdot, t)$ matches the prescribed initial quantity $\int_\Omega h(x)\,\mathrm{d}x$ for all t.

To summarize, post-processing is integrated into FNO-PINN as follows. During Stage 1 and Stage 2, the objectives J_{op} and J_{fine} are minimized with respect to the unmodified network output u_θ. The discrete operator N_h and all loss terms are as defined in Section 3.4, without any post-processing. After Stage 2 has converged, we apply one or more of the above operators (3.59)–(3.62) to obtain the final prediction $\widehat{u_\theta}$, which is used for reporting error metrics and visualizing solution fields in Section 4. Optionally, the same post-processing can also be applied to the Stage 1 FNO output when we compare different variants (e.g., pure FNO vs. FNO-PINN).

In this way, post-processing is a simple correction at inference. It enforces exact boundary/initial conditions, periodicity, and conservation. The training procedure stays fully differentiable and follows the loss formulations in Section 3.

3.7. Dynamic weight adjustment strategy

In multi-component PDE systems, different components change at different speeds. With fixed weights, the optimization focuses on components with larger gradients. These are usually the fast-changing ones, so the slow components become less accurate. To solve this problem, we propose a dynamic weight adjustment strategy. It balances different loss components based on how fast they decrease during training.

Let $L_m^{(e)}$ denote the loss for component $m \in \{\text{PDE}, \text{IC}\}$ at epoch e. At each update step, we compute the moving-window averages over a window of size W:

$$\bar{L}_m^{\text{old}} = \frac{2}{W} \sum_{i=e-W}^{e-W/2-1} L_m^{(i)}, \quad \bar{L}_m^{\text{new}} = \frac{2}{W} \sum_{i=e-W/2}^{e-1} L_m^{(i)}. \tag{3.63}$$

The descent rate for component m is then defined as

$$r_m = \frac{\bar{L}_m^{\text{old}} - \bar{L}_m^{\text{new}}}{\bar{L}_m^{\text{old}} + \epsilon}, \tag{3.64}$$

where $\epsilon = 10^{-10}$ is a small constant to prevent division by zero. A positive r_m indicates that the loss is decreasing, with larger values indicating faster descent.

The weights are updated every P epochs (after a warmup period) according to the following rules. If $r_{\text{PDE}} < \gamma_1 \cdot r_{\text{IC}}$ (PDE loss descends slower than IC loss):

$$\lambda_{\text{PDE}} \leftarrow \min(\lambda_{\text{PDE}} \cdot (1 + \alpha),\ \lambda_{\text{PDE}}^{\max}). \tag{3.65}$$

If $r_{\text{IC}} < \gamma_2 \cdot r_{\text{PDE}}$ (IC loss descends slower than PDE loss):

$$\lambda_{\text{IC}} \leftarrow \min(\lambda_{\text{IC}} \cdot (1 + \alpha),\ \lambda_{\text{IC}}^{\max}). \tag{3.66}$$

The motivation behind this strategy is as follows: if a loss component is descending slower than others relative to the threshold, it is receiving insufficient optimization attention, and its weight should be increased. The update factor $(1 + \alpha)$ makes changes gradual. The upper bounds $\lambda_m^{\max}$ stop any single component from becoming too large.

The following parameter values are used in all experiments:

Adjustment factor: $\alpha = 0.1$. Threshold coefficients: $\gamma_1 = 0.8$, $\gamma_2 = 0.5$. Window size: $W = 20$ epochs. Update period: $P = 10$ epochs. Warmup period: 30 epochs. Initial weights: $\lambda_{\text{PDE}}^{(0)} = 1.0$, $\lambda_{\text{IC}}^{(0)} = 10.0$. Upper bounds: $\lambda_{\text{PDE}}^{\max} = 5.0$, $\lambda_{\text{IC}}^{\max} = 50.0$. Initial weights: $\lambda_{\text{PDE}}^{(0)} = 1.0$, $\lambda_{\text{IC}}^{(0)} = 10.0$.

The adjustment factor $\alpha = 0.1$ ensures gradual weight changes, avoiding oscillatory behavior that can occur with larger values. The asymmetric thresholds ($\gamma_1 = 0.8 > \gamma_2 = 0.5$) reflect the observation that IC loss typically converges faster in the early training stages and can afford more relaxed monitoring. The window size $W = 20$ provides sufficient smoothing of epoch-to-epoch fluctuations while remaining responsive to genuine trends in the loss landscape. These parameter choices were validated through preliminary experiments, showing stable convergence across different problem types.

4. Numerical simulation

To test the FNO-PINN method, we design typical examples.

All experiments were conducted on a workstation equipped with an Intel Core i9-10900K CPU (3.7 GHz, 10 cores) with 64 GB of RAM. The implementation uses PyTorch with Python 3.x, and all computations were performed on CPU.

Training data specifications. Our FNO-PINN framework learns a coordinate-to-solution mapping $(x, t) \mapsto u(x, t)$ for fixed parameter values rather than a parameter-to-solution mapping that would require sampling parameters from a distribution. Table 2 summarizes the fixed parameters and training data specifications for each example.

Table 2. Fixed parameter values and training data specifications for each numerical example.

Example	Fixed Parameters	Grid Size	Training Samples
Nonlinear coupled system	$b = 0.1$, $\delta = 0.1$	128×64	8192
Coupled diffusion system	$\beta = 1.0$, $D_{11} = D_{22} = 1$, $D_{12} = D_{21} = 0$	128×64	8192
2D Burgers equation	$\nu = 0.01$	$32 \times 32 \times 16$	16,384

Table 3 reports the computational time for each phase of training.

Table 3. Computational cost breakdown for FNO-PINN training (CPU, Intel i9-10900K).

Example	Stage 1 (s)	Stage 2 (s)	Total (s)	Stage 1 Ratio
Nonlinear coupled	151.25	151.69	302.93	49.9%
Coupled diffusion	343.20	320.83	664.03	51.7%
2D Burgers	1115.36	968.68	2084.04	53.5%

The Stage 1 pre-training accounts for approximately 50–54% of total training time. This overhead is justified by the improved accuracy, as FNO-learned frequency-domain features provide effective initialization that helps Stage 2 optimization avoid poor local minima.

For both PINNs and FNO-PINN, we use the same training settings for fair comparison. All examples use Adam optimizer with initial learning rate 5×10^{-4} and cosine annealing schedule Eq (3.53) and with 2500 epochs for Stage 1 and 2500 epochs for Stage 2. Gradient clipping with

$\gamma = 1.0$ and L_2 regularization with $\lambda_{\text{reg}} = 10^{-6}$ is applied in both phases. The batch size is 2048 for all experiments.

In all numerical examples (Sections 4.1–4.3), both the operator-learning phase (Stage 1) and the instance-specific fine-tuning phase (Stage 2) are trained on the full space–time domain using the same uniform tensor grid.

4.1. Nonlinear coupled equations

$$\begin{cases} \dfrac{\partial u_0}{\partial t} = \dfrac{\partial^2 u_0}{\partial x^2} + \dfrac{\partial^2 U_1}{\partial x^2} + 2bu_0 + 2\delta u_1, \\ \dfrac{\partial u_1}{\partial t} = \dfrac{\partial^2 u_1}{\partial x^2} + \dfrac{\partial^2 U_2}{\partial x^2} + 2bu_1 + 2\delta(u_0 + 2u_2), \\ \dfrac{\partial u_2}{\partial t} = \dfrac{\partial^2 u_2}{\partial x^2} + \dfrac{\partial^2 U_3}{\partial x^2} + 2bu_2 + 2\delta u_1 \end{cases} \tag{4.1}$$

where the nonlinear functions U_1, U_2, U_3 are defined as follows:

$$\begin{cases} U_1 = \dfrac{1}{3}(u_0^3 + 3u_0u_1^2 + 3u_0u_2^2 + 3u_1^2u_2 + u_2^3), \\ U_2 = \dfrac{1}{3}(3u_0^2u_1 + 6u_0u_1u_2 + 2u_1^3 + 6u_1u_2^2), \\ U_3 = \dfrac{1}{3}(3u_0^2u_2 + 3u_0u_1^2 + 3u_0u_2^2 + 6u_1^2u_2 + 3u_2^3). \end{cases} \tag{4.2}$$

The boundary conditions are homogeneous Dirichlet conditions as follows:

$$u_0(x_{\min}, t) = u_0(x_{\max}, t) = u_1(x_{\min}, t) = u_1(x_{\max}, t) = u_2(x_{\min}, t) = u_2(x_{\max}, t) = 0.$$

The initial conditions are as follows:

$$u_0(x, 0) = e^{-x^2}, \quad u_1(x, 0) = 0, \quad u_2(x, 0) = 0.$$

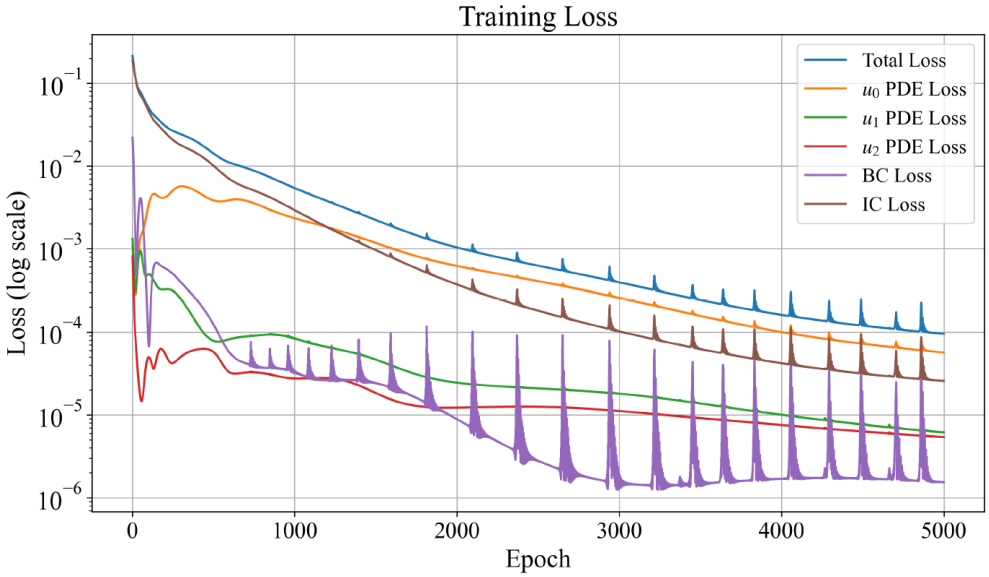
The computational domain is $x \in [-5, 5]$, $t \in [0, 1]$.

In this nonlinear coupled system, we set $b = 0.1$ and $\delta = 0.1$. A single FNO-PINN network outputs all three components (u_0, u_1, u_2). We do not train separate networks for each component. In Stage 1, we train this network on the global grid $x \in [-5, 5]$, $t \in [0, 1]$. It learns an operator that maps parameters and initial data to (u_0, u_1, u_2). We use the loss J_{op} from Section 3.6.3 with fixed weights. In Stage 2, we fix the parameters (b, δ) and initial conditions. We initialize the network from Stage 1 and continue training on the same grid. We use the fine-tuning loss J_{fine} from Section 3.6.4. In this phase, the dynamic weight strategy balances the PDE and IC terms for the three components. These components have different temporal behaviors. The anchor loss keeps the solution close to the Stage 1 operator.

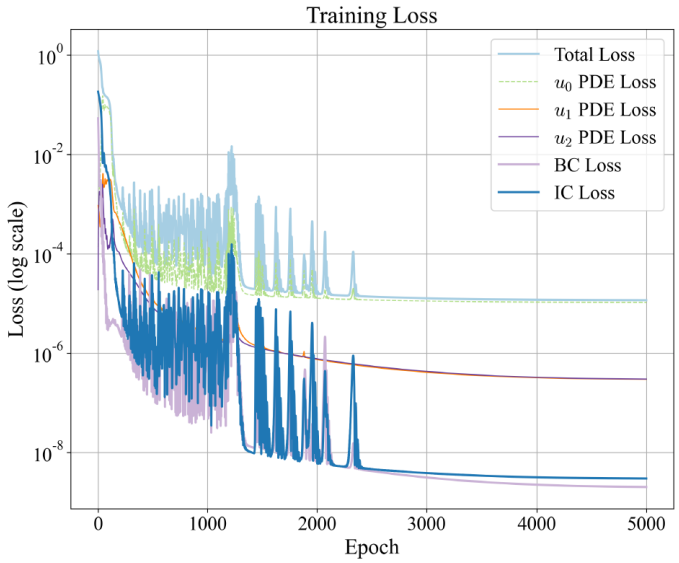
To comprehensively evaluate the performance of the three methods, we use the results of the FD method as the benchmark and calculate the relative errors of Eq (4.1).

Figure 5 show the loss function of the two methods after 5000 training times. From Figure 5(b), it is clear that the FNO-PINN not only converges faster but also reaches a lower loss function value in the end. This indicates that the FNO-PINN is able to learn the dynamics of the system more effectively, especially when dealing with multiple timescales coupled systems. The loss function of PINNs still

presents large fluctuations in the late training period, which reflects the training stability challenge faced by standard PINNs when dealing with systems with different time scales.



(a) PINN loss convergence



(b) FNO-PINN loss convergence

Figure 5. Comparison of loss convergence curves for PINNs and FNO-PINN

In Table 4, when solving the equivalent deterministic equations, the advantages of FNO-PINN become more pronounced. The mean root mean square error (RMSE) has decreased to 1.303×10^{-4} (compared to the random system, it has decreased by 85.9%), and the L2 error is only 1.179×10^{-2}, demonstrating the ability to fully utilize the deterministic features. Although the mean RMSE of PINNs has improved (6.399×10^{-4}), the L2 error has increased abnormally (4.592×10^{-1}), revealing the limitations of handling high-dimensional equivalent systems. The MC method still performs poorly, indicating that it is not suitable for complex deterministic systems. Combining the results of the two

tables, FNO-PINN maintains the optimal performance under different problem settings, proving the stability and superiority of this method. To quantify the prediction accuracy, we define the following error metrics. For a system with m solution components u_i ($i = 1, 2, \ldots, m$), let u_i^{pred} and u_i^{ref} denote the predicted and reference solutions, respectively, evaluated at N discrete points.

Table 4. Error indicators of different methods relative to Eq (4.1).

Method	Mean RMSE	Mean relative RMSE	Mean L2 error	Max error
MC	1.746×10^{-3}	3.665×10^{-2}	1.253×10^{-1}	2.265×10^{-2}
PINNs	6.399×10^{-4}	4.261×10^{-2}	4.592×10^{-1}	3.911×10^{-3}
FNO-PINN	1.303×10^{-4}	6.753×10^{-3}	1.179×10^{-2}	2.175×10^{-3}

The RMSE for component i is defined as

$$\text{RMSE}_i = \sqrt{\frac{1}{N}\sum_{j=1}^{N}\left(u_i^{\text{pred}}(x_j, t_j) - u_i^{\text{ref}}(x_j, t_j)\right)^2}.$$

The mean RMSE is the average over all components:

$$\text{Mean RMSE} = \frac{1}{m}\sum_{i=1}^{m}\text{RMSE}_i.$$

The relative RMSE for component i is

$$\text{Relative RMSE}_i = \frac{\text{RMSE}_i}{\sqrt{\frac{1}{N}\sum_{j=1}^{N}\left(u_i^{\text{ref}}(x_j, t_j)\right)^2}}.$$

The mean relative RMSE is the average over all components:

$$\text{Mean relative RMSE} = \frac{1}{m}\sum_{i=1}^{m}\text{Relative RMSE}_i.$$

The L2 error for component i is defined as

$$\text{L2 error}_i = \left\|u_i^{\text{pred}} - u_i^{\text{ref}}\right\|_{L^2} \approx \sqrt{\sum_{j=1}^{N}\left(u_i^{\text{pred}}(x_j, t_j) - u_i^{\text{ref}}(x_j, t_j)\right)^2 \Delta x \Delta t}.$$

The Mean L2 error is the average over all components:

$$\text{Mean L2 error} = \frac{1}{m}\sum_{i=1}^{m}\text{L2 error}_i.$$

The max error is the maximum pointwise absolute error:

$$\text{Max Error} = \max_{i,j}\left|u_i^{\text{pred}}(x_j, t_j) - u_i^{\text{ref}}(x_j, t_j)\right|.$$

Table 5 compares the performance of AD and FD methods under the same static weight configuration. The results demonstrate that the FD method provides a dramatic improvement in solution accuracy. Specifically, the total RMSE is reduced from 8.154×10^{-2} to 1.664×10^{-3}, representing an improvement of 97.96% (nearly 49 times more accurate). At the initial time t_0, the RMSE is reduced by 98.94%, at intermediate time t_1 by 97.76%, and at final time t_2 by 93.26%.

Table 5. The influence of the derivative calculation method of Eq (4.1) under static weights.

Method	Total RMSE	t_0 RMSE	t_1 RMSE	t_2 RMSE
AD + Static Weights	8.154×10^{-2}	1.764×10^{-1}	3.271×10^{-2}	2.775×10^{-3}
FD + Static Weights	1.664×10^{-3}	1.871×10^{-3}	7.335×10^{-4}	1.871×10^{-4}

This big improvement shows that the high-order finite difference scheme gives more accurate derivatives. It works well for both linear diffusion terms $\frac{\partial^2 u_i}{\partial x^2}$ and nonlinear diffusion terms $\frac{\partial^2 U_i}{\partial x^2}$. Here U_1, U_2, U_3 are the cubic coupling functions in Eq (4.1). Accurate second-order spatial derivatives are important for capturing the complex nonlinear coupling in this three-component system.

For this nonlinear coupled system, we only show the AD vs. FD comparison (Table 5). We do not include a separate comparison of static and dynamic weight strategies. We make this choice for the following reasons:

Dominant FD contribution: as shown in Table 5, the FD method alone gives a 97.96% improvement in accuracy. With such a large effect, the contribution of dynamic weights is small and hard to measure separately.

Complex nonlinear coupling: the three components (u_0, u_1, u_2) are coupled through nonlinear functions U_1, U_2, U_3. These involve cubic and cross terms. In the coupled diffusion system, the timescale separation is clear: $e^{-4\beta t}$ vs. $e^{-\beta t}$. Here, the multi-scale behavior comes from complex nonlinear interactions, so the "attention shift" problem is less clear.

Controlled validation: the coupled diffusion system in Section 4.2 is better for testing the dynamic weight strategy. It has linear dynamics and clearly separated timescales.

For a clear test of the dynamic weight strategy, see Table 8 in Section 4.2.

Figures 6 and 7 respectively show the spatial and temporal distribution of the predicted solutions by these two methods. It can be observed that FNO-PINN can more accurately capture the dynamic behavior of the system at different timescales, especially demonstrating a significant advantage in the prediction of the u_1 and u_2 components. This is mainly due to the fact that FNO-PINN combines the global feature extraction capability of FNOs, the physical constraint mechanism of PINNs, and the multi-objective loss function and dynamic weight adjustment strategy we proposed.

It is worth noting that in this nonlinear coupled system, the u_0 component shows a faster attenuation characteristic, while the u_1 and u_2 components exhibit more complex time-varying evolution patterns. When dealing with such multi-timescale problems, traditional PINNs are prone to the attention shift issue, which is due to the significant numerical differences among different components. The network tends to optimize the components with larger numerical values and neglects the accuracy of the smaller ones. However, FNO-PINN effectively alleviate this problem through time-adaptive weights and batch normalization, demonstrating the effectiveness of the method proposed in this paper.

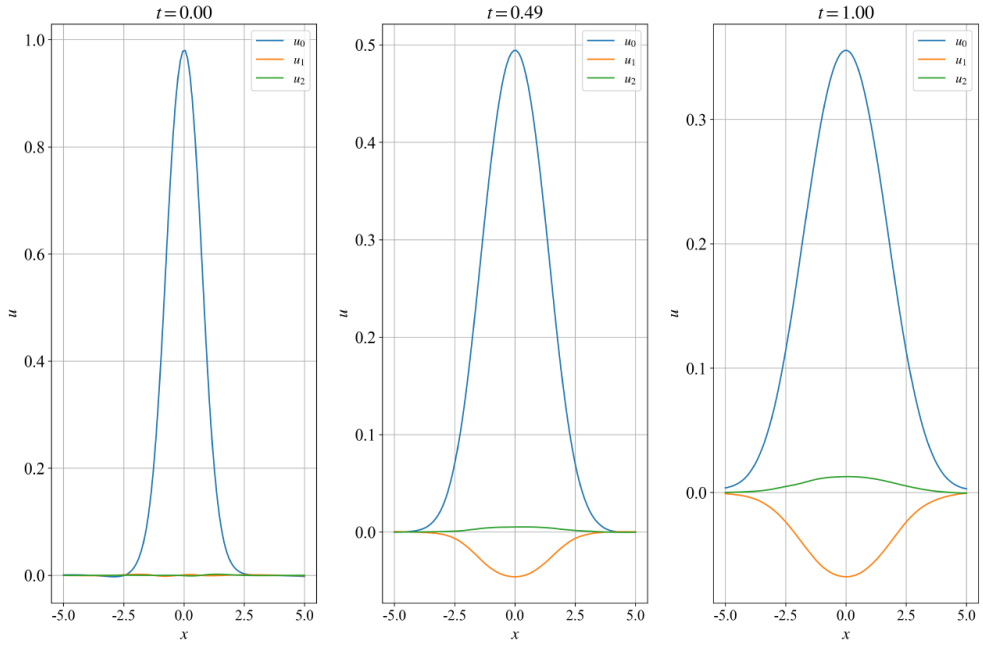


Figure 6. The predicted values using PINN.

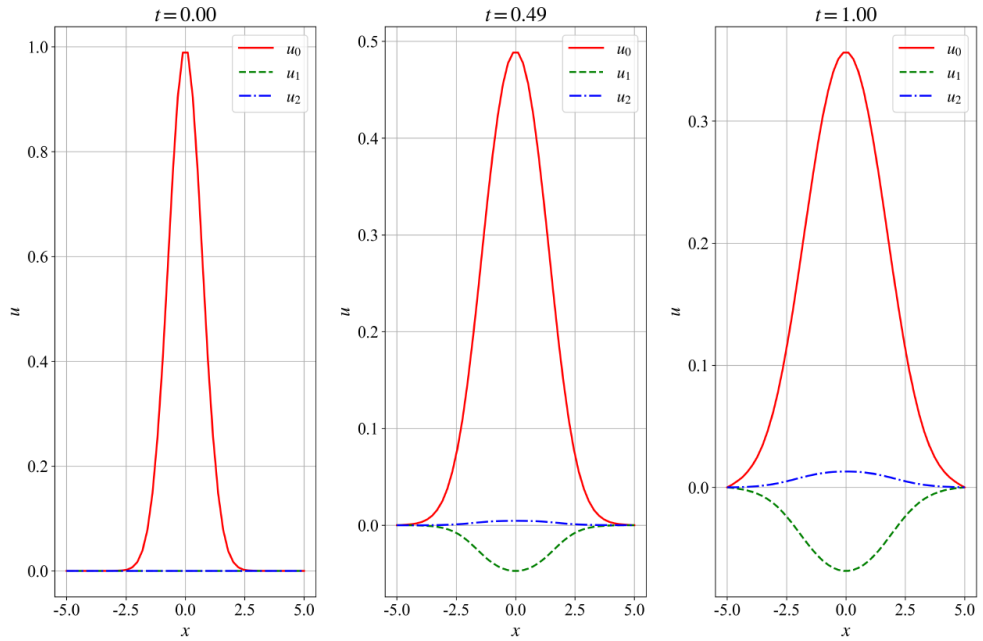


Figure 7. The predicted values using FNO-PINN.

4.2. Coupled diffusion system

The second example considers a coupled diffusion system with periodic boundary conditions as follows:

$$\begin{cases} \dfrac{\partial u}{\partial t} = D_{11}\dfrac{\partial^2 u}{\partial x^2} + D_{12}\dfrac{\partial^2 v}{\partial x^2}, & x \in [0, 2\pi],\ t > 0, \\ \dfrac{\partial v}{\partial t} = D_{21}\dfrac{\partial^2 u}{\partial x^2} + D_{22}\dfrac{\partial^2 v}{\partial x^2}, & x \in [0, 2\pi],\ t > 0. \end{cases} \tag{4.3}$$

The initial conditions are as follows:

$$u(x, 0) = \cos(2x), \quad v(x, 0) = \cos(x). \tag{4.4}$$

The boundary conditions are periodic boundary conditions

$$\begin{aligned} &u(0, t) = u(2\pi, t), \quad v(0, t) = v(2\pi, t), \\ &\left.\frac{\partial u}{\partial x}\right|_{x=0} = \left.\frac{\partial u}{\partial x}\right|_{x=2\pi}, \quad \left.\frac{\partial v}{\partial x}\right|_{x=0} = \left.\frac{\partial v}{\partial x}\right|_{x=2\pi}. \end{aligned} \tag{4.5}$$

The exact solution of this problem is as follows:

$$u(x, t) = e^{-4\beta t}\cos(2x), \quad v(x, t) = e^{-\beta t}\cos(x). \tag{4.6}$$

In this example, we set the diffusion coefficients as $D_{11} = D_{22} = 1.0$ and $D_{12} = D_{21} = 0$, with $\beta = 1.0$. This example is special in that the two components have significantly different timescales. The u component decays 4 times faster than the v component, making it an ideal test case to verify the ability of our method to handle multi-timescales problems.

In this coupled system, the two variables (u, v) are also represented by a single FNO-PINN network with a two-dimensional output. We do not treat different variables separately in the network architecture. The FD-based residuals and dynamic weight strategy handle the different timescales. Like Section 4.1, Stage 1 learns an operator on the full grid $x \in [0, 2\pi]$, $t \in [0, 1]$ with fixed weights. Stage 2 fine-tunes the same network for specific diffusion coefficients and initial data. The dynamic weights balance the PDE and IC losses of u and v. This ensures neither component is ignored during training.

The grid resolution is 128 points in the spatial direction and 64 points in the temporal direction. The other training parameters are the same as in the first example. These include the optimizer, learning rate schedule, and total epochs. Both stages run on the same global space-time grid.

From Table 6, it can be seen that the MC method performs poorly in all error indicators and is unable to effectively capture the fine structure of the solution over multiple time scales. PINN outperforms MC in relative error, but its overall accuracy is still limited, and the maximum error is relatively large, indicating that it has training instability in this type of problem. In contrast, the errors of FNO-PINN are significantly lower than those of the former two methods. The mean square error and $L2$ error reach the 10^{-4} magnitude, and the relative error is reduced by more than two orders of magnitude, indicating that this method has obvious advantages in handling coupled diffusion equations and different timescale attenuation characteristics.

Table 6. The error indicators of different methods for Eq (4.3).

Method	Mean RMSE	Mean relative RMSE	Mean $L2$ error	Max Error
MC	1.126×10^{-1}	5.687×10^{-1}	3.572×10^{-1}	2.579×10^{-2}
PINNs	1.430×10^{-1}	2.885×10^{-1}	6.576×10^{-1}	2.516×10^{-1}
FNO-PINN	1.246×10^{-4}	5.485×10^{-3}	4.852×10^{-4}	5.358×10^{-4}

Table 7 compares the performance of AD and FD methods under the same static weight configuration. The results demonstrate that the FD method provides substantial improvement in solution accuracy for the coupled diffusion system. Specifically, the total RMSE is reduced from 1.618×10^{-1} to 5.073×10^{-2}, representing an improvement of 68.65% (approximately 3.2 times more accurate). For the fast-decaying component u, the RMSE is reduced by 75.97% (from 1.522×10^{-1} to 3.657×10^{-2}); for the slow-decaying component v, the RMSE is reduced by 36.00% (from 5.494×10^{-2} to 3.516×10^{-2}). The bigger improvement in u is because u decays 4 times faster than v. It needs more accurate derivatives to capture its fast changes. The high-order finite difference scheme computes the coupled diffusion terms $D_{12}\frac{\partial^2 v}{\partial x^2}$ and $D_{21}\frac{\partial^2 u}{\partial x^2}$ more accurately. This is important for keeping the physical consistency of the multi-scale solution.

Table 7. The influence of the derivative calculation method of Eq (4.3) under static weights.

Method	RMSE(u)	RMSE(v)	Total RMSE
AD + Static Weights	1.522×10^{-1}	5.494×10^{-2}	1.618×10^{-1}
FD + Static Weights	3.657×10^{-2}	3.516×10^{-2}	5.073×10^{-2}

The ablation experiments in Tables 5 and 7 show that FD-based derivatives are much better than AD. The improvement is 97.96% for the nonlinear coupled system and 68.65% for the coupled diffusion system. These results support our choice in Section 3. The high-order FD scheme is needed for high accuracy in FNO-PINN.

Table 8 compares the performance of static and dynamic weight strategies under the FD method. The coupled diffusion system demonstrates significant benefits from the dynamic weight adjustment: the total RMSE is reduced from 5.073×10^{-2} to 3.059×10^{-2}, representing an improvement of 39.71%. For the fast-decaying component u, the RMSE decreases by 48.71% (from 3.657×10^{-2} to 1.876×10^{-2}); for the slow-decaying component v, the reduction is 31.27% (from 3.516×10^{-2} to 2.417×10^{-2}).

Table 8. The influence of the weight strategy of Eq (4.3) under the FD method.

Method	RMSE(u)	RMSE(v)	Total RMSE
FD + Static Weights	3.657×10^{-2}	3.516×10^{-2}	5.073×10^{-2}
FD + Dynamic Weights	1.876×10^{-2}	2.417×10^{-2}	3.059×10^{-2}
Improvement	48.71%	31.27%	39.71%

This big improvement can be explained by the "attention shift" problem in multi-scale systems. The component u decays at rate $e^{-4\beta t}$, which is 4 times faster than v at $e^{-\beta t}$. With static weights, u dominates the loss gradients. This is because $\partial u/\partial t$ has larger magnitude. Also, the PDE residual for u contributes too much to the total loss in early training, so the network focuses on u and does not optimize v well.

The dynamic weight strategy (Section 3.7) fixes this imbalance. It monitors how fast different loss components decrease. When a loss component decreases slower than expected, its weight increases automatically. This ensures balanced optimization for both time scales. The improvement is larger for u (48.71%) than for v (31.27%). Dynamic weighting prevents over-optimization of u. This helps the network achieve a balanced solution with good accuracy for both components.

From the comparison of loss functions in Figures 8 and 9, it can be seen that the training process of FNO-PINN is more stable and converges faster. The loss function of PINNs oscillates significantly during training, indicating that its optimization process is unstable and prone to fall into local optima or saddle points.

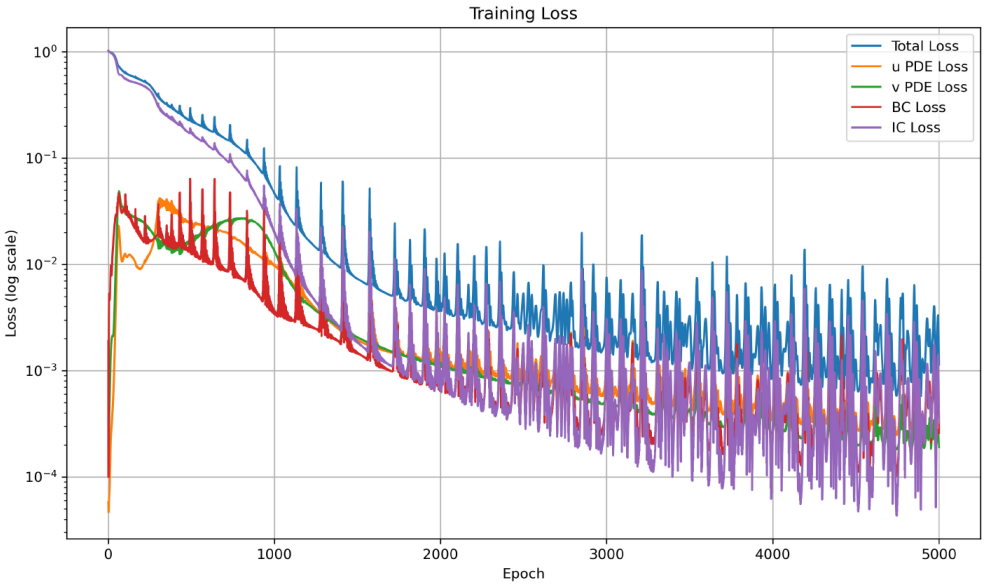


Figure 8. The loss convergence curve trained by the PINN method.

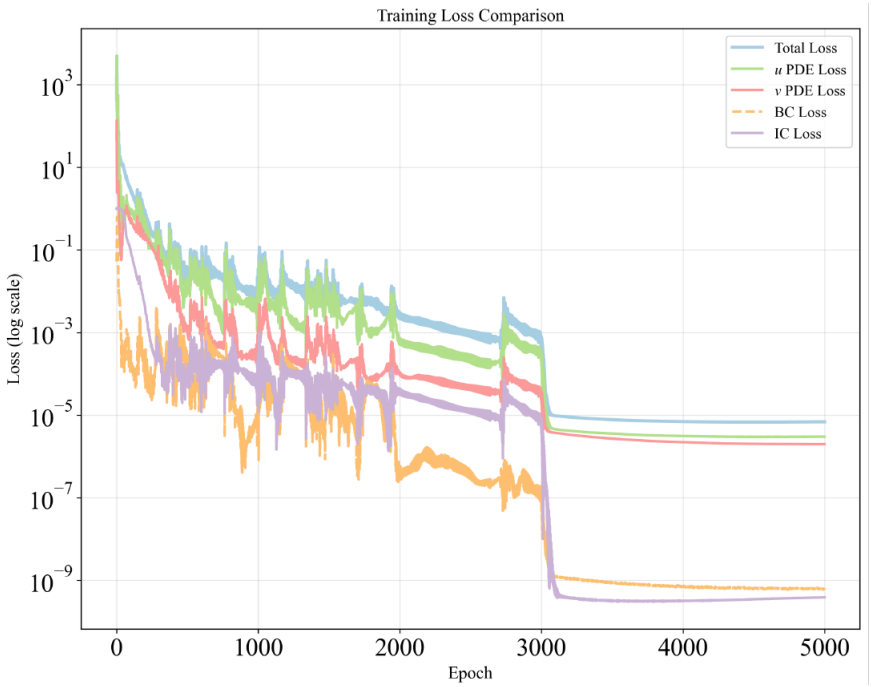


Figure 9. The loss convergence curve trained by the FNO-PINN method.

Figures 10 and 11 show the accurate value and error diagram of u and v generated by using traditional PINNs. It can be seen that the error is large, and the fitting effect is poor.

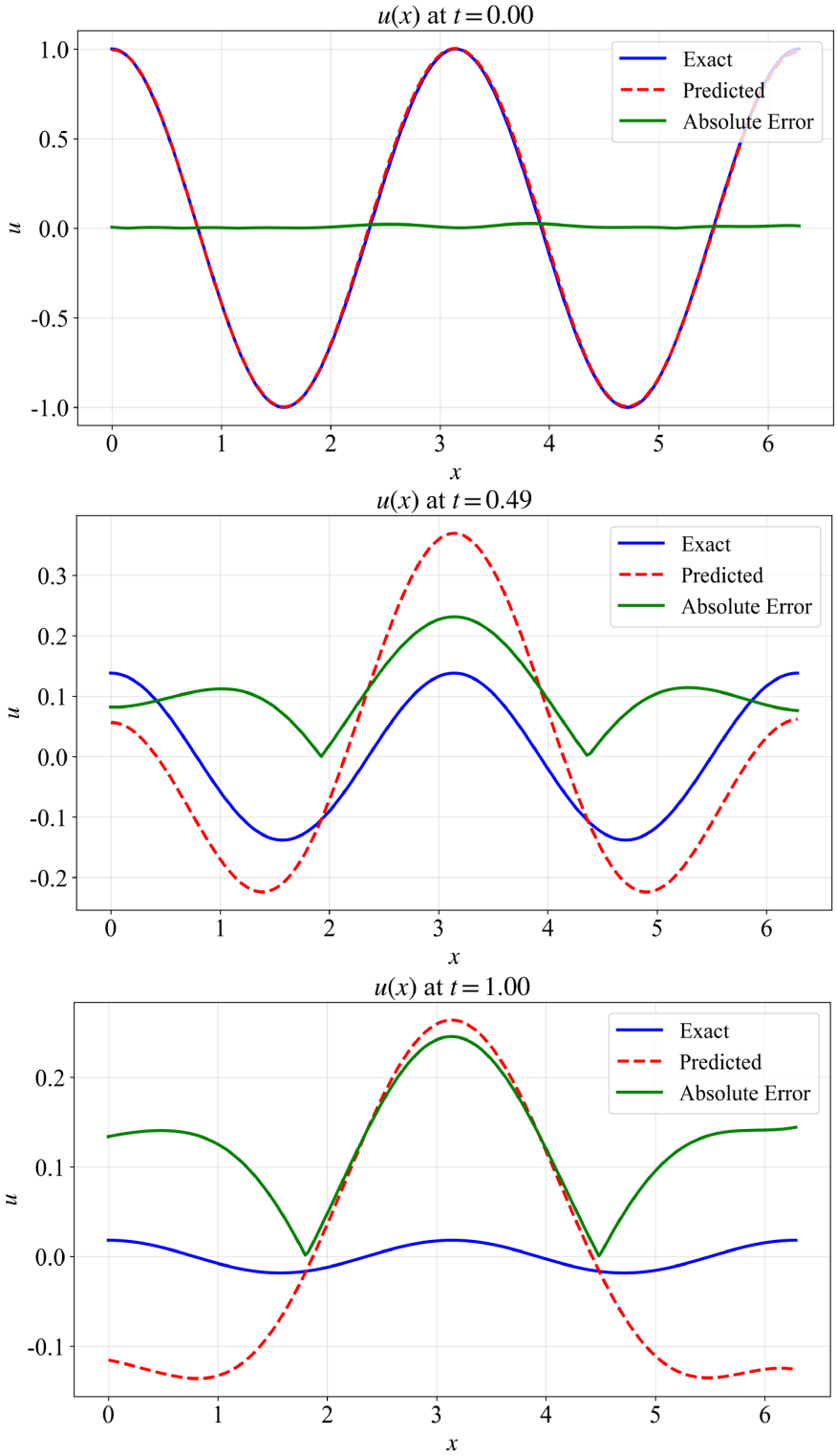


Figure 10. Comparison of the predicted value for variable u obtained by PINNs and the exact solution.

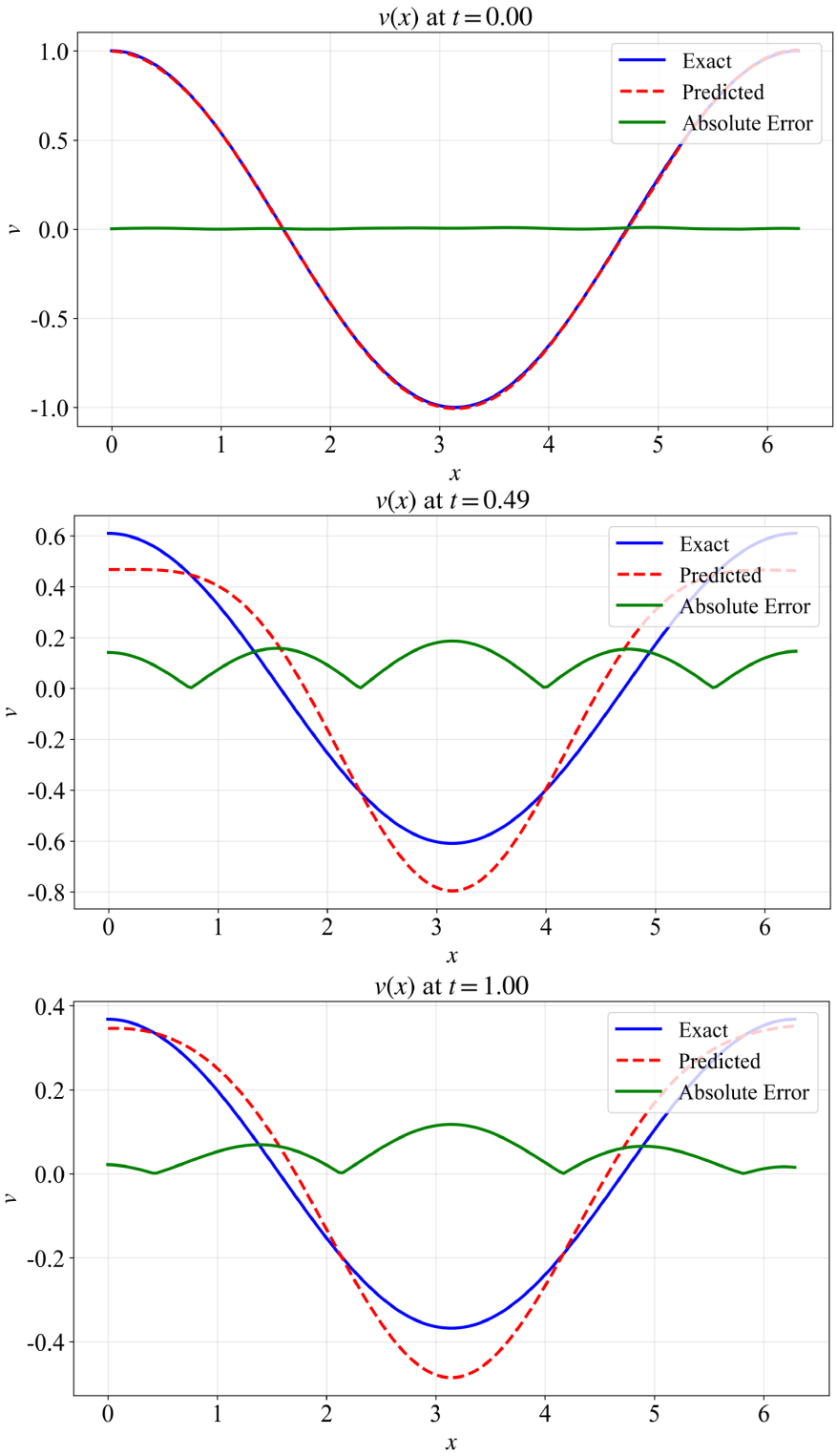


Figure 11. Comparison of the predicted value for variable v obtained by PINN and the exact solution.

Figures 12 and 13 show the accurate value of u and v generated by using FNO-PINN and the error graph. It can be seen that the error is very small, and the fitting effect is very good.

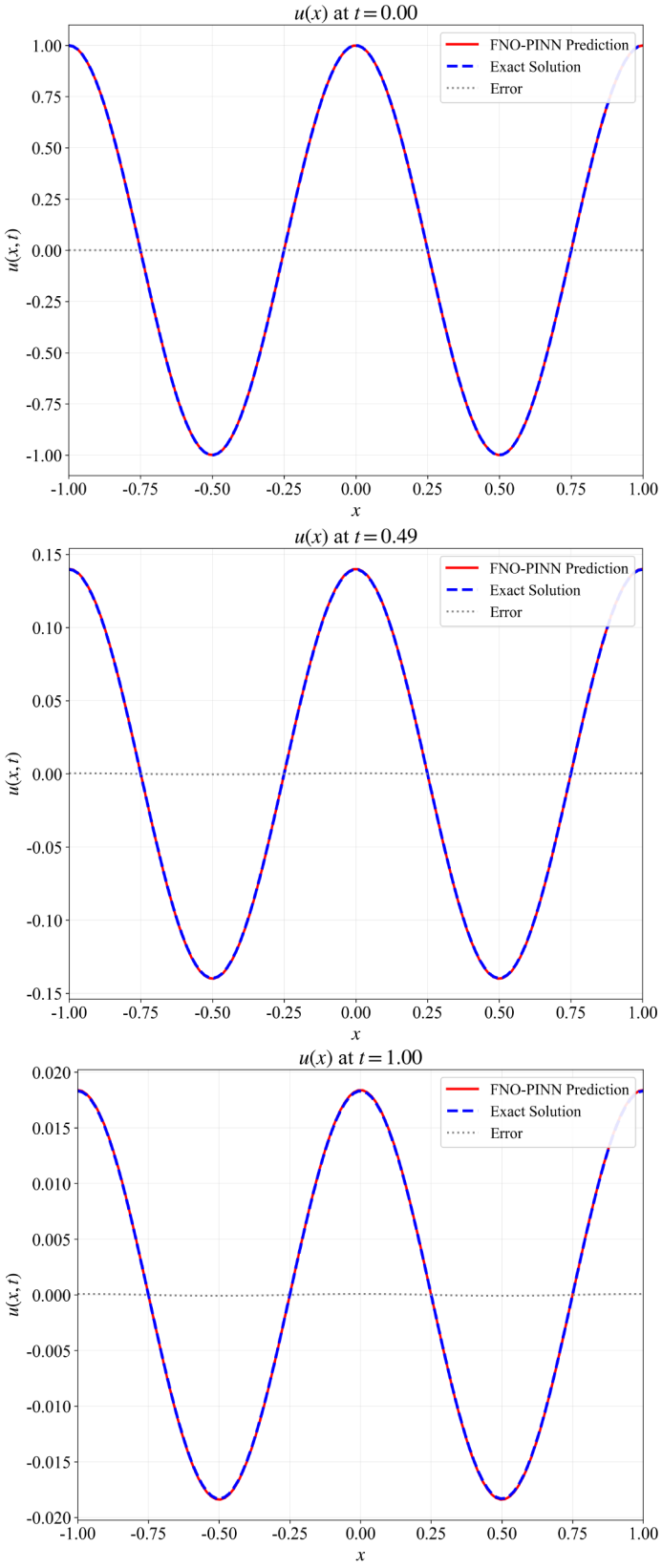


Figure 12. Comparison of the predicted value for variable u obtained by FNO-PINN and the exact solution.

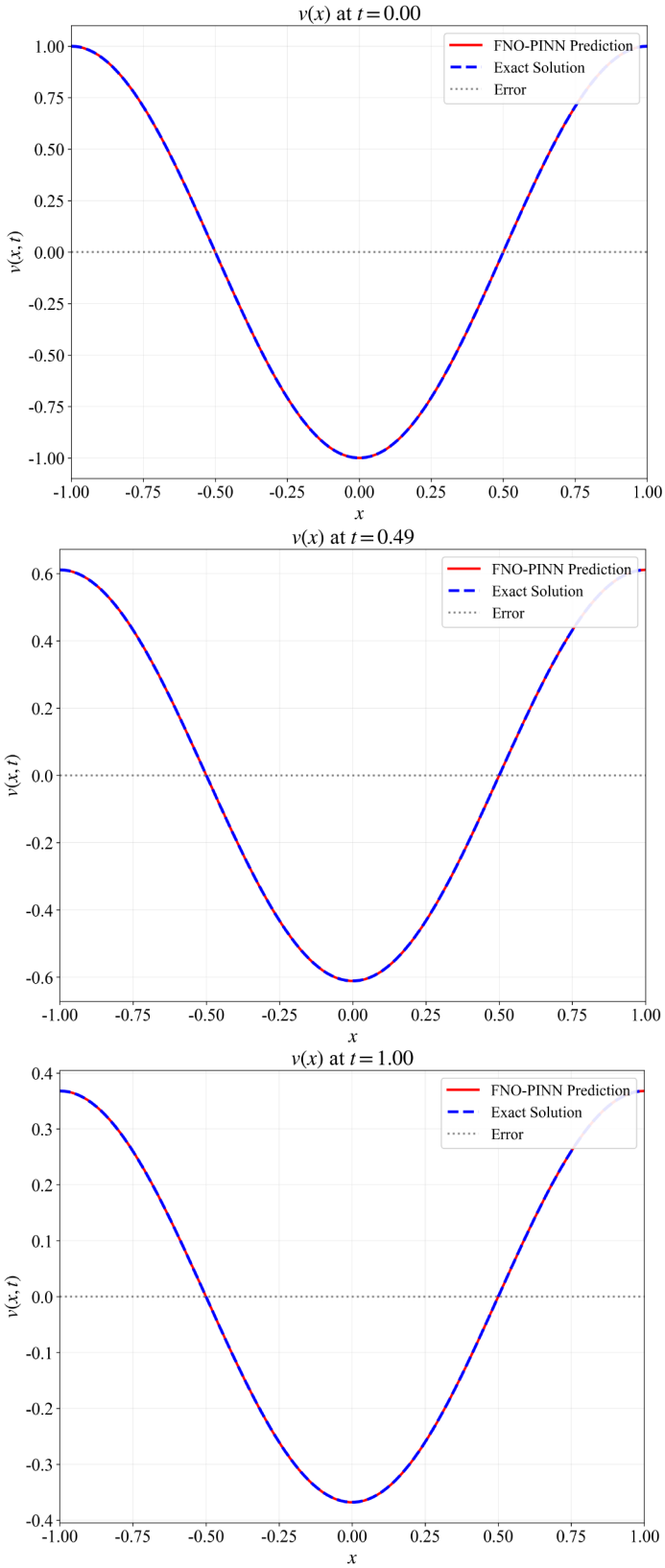


Figure 13. Comparison of the predicted value for variable v obtained by FNO-PINN and the exact solution.

The above results clearly demonstrate the differences between the two methods in the comparison of predicted solutions and exact solutions, as well as the corresponding error distribution. It can be seen that the predicted results of FNO-PINN are almost completely consistent with the exact solution, with extremely small errors and a concentrated distribution. In contrast, the prediction deviation of traditional PINNs is larger, and the error distribution is more dispersed, indicating that it has certain limitations when depicting multi-timescale dynamic characteristics.

4.3. Two-dimensional Burgers' equation

The third example considers a two-dimensional Burgers' equation system with Dirichlet boundary conditions:

$$\begin{cases} \dfrac{\partial u}{\partial t} + u\dfrac{\partial u}{\partial x} + v\dfrac{\partial u}{\partial y} = \nu\left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2}\right), & (x, y) \in [0, 1]^2,\ t > 0, \\ \dfrac{\partial v}{\partial t} + u\dfrac{\partial v}{\partial x} + v\dfrac{\partial v}{\partial y} = \nu\left(\dfrac{\partial^2 v}{\partial x^2} + \dfrac{\partial^2 v}{\partial y^2}\right), & (x, y) \in [0, 1]^2,\ t > 0. \end{cases} \tag{4.7}$$

The initial conditions are:

$$u(x, y, 0) = \sin(\pi x)\sin(\pi y), \quad v(x, y, 0) = \sin(\pi x)\sin(\pi y). \tag{4.8}$$

The boundary conditions are homogeneous Dirichlet,

$$u(x, y, t)|_{\partial\Omega} = 0, \quad v(x, y, t)|_{\partial\Omega} = 0, \tag{4.9}$$

where $\partial\Omega$ is the boundary of the domain $[0, 1]^2$.

We set the viscosity coefficient as $\nu = 0.01$ and the time domain as $t \in [0, 0.5]$. The two-dimensional Burgers' equation is a classical nonlinear convection-diffusion equation. It shows both diffusive smoothing and nonlinear convective transport. This leads to steep gradients and asymmetric solution profiles. This makes it a good test case for our method's ability to handle nonlinear PDEs with strong convection.

A single FNO-PINN network outputs both velocity components (u, v). The nonlinear convection terms like $u\partial u/\partial x$ and $v\partial u/\partial y$ are hard for traditional methods due to the strong coupling between variables. The grid resolution is 32×32 points in the spatial directions and 16 points in the temporal direction.

Table 9 shows that the MC method performs poorly in all error metrics. Its mean RMSE is 4.565×10^{-2} and maximum error is 6.984×10^{-1}. MC cannot capture the nonlinear dynamics of the Burgers' equation well.

Table 9. The error indicators of different methods for Eq (4.7).

Method	Mean RMSE	Mean relative RMSE	Mean $L2$ error	Max error
MC	4.565×10^{-2}	1.060×10^{-1}	1.060×10^{-1}	6.984×10^{-1}
PINNs	4.210×10^{-2}	9.777×10^{-2}	9.777×10^{-2}	4.061×10^{-1}
FNO-PINN	1.940×10^{-4}	4.506×10^{-4}	4.506×10^{-4}	1.905×10^{-3}

PINN performs slightly better than MC with a mean RMSE of 4.210×10^{-2}, but its maximum error of 4.061×10^{-1} shows big local prediction failures. This suggests training instability for nonlinear convection-dominated problems.

In contrast, FNO-PINN errors are much lower. The mean RMSE is 1.940×10^{-4}, more than two orders of magnitude smaller than MC and PINN. The relative RMSE and $L2$ error are both 4.506×10^{-4}. The maximum error is only 1.905×10^{-3}. FNO-PINN handles the two-dimensional nonlinear Burgers' equation with strong convection very well.

Figures 14 and 15 compare the loss functions. FNO-PINN training shows a periodic pattern with rapid convergence after each cycle. The final loss reaches 10^{-7} to 10^{-8}. In contrast, the PINN loss function shows big oscillations during training. The final loss only reaches 10^{-3} to 10^{-4}. This shows that PINN optimization is unstable and may get stuck in local optima when handling nonlinear convection terms.

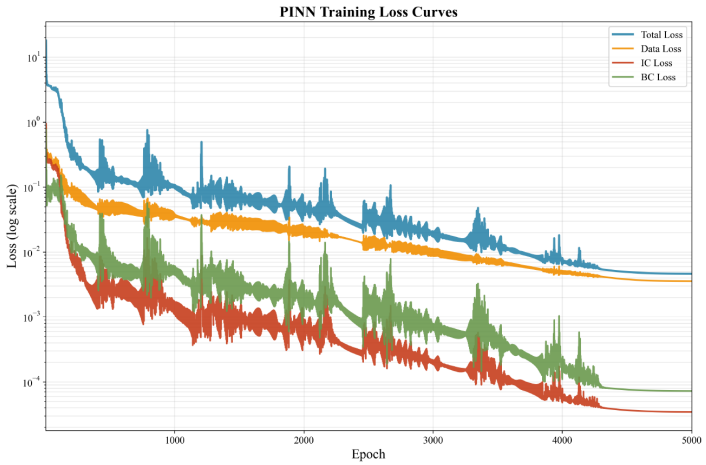


Figure 14. The loss convergence curve trained by the PINN method.

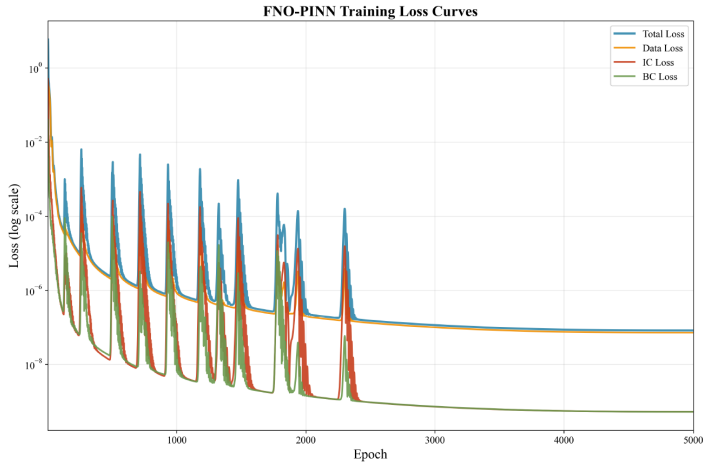
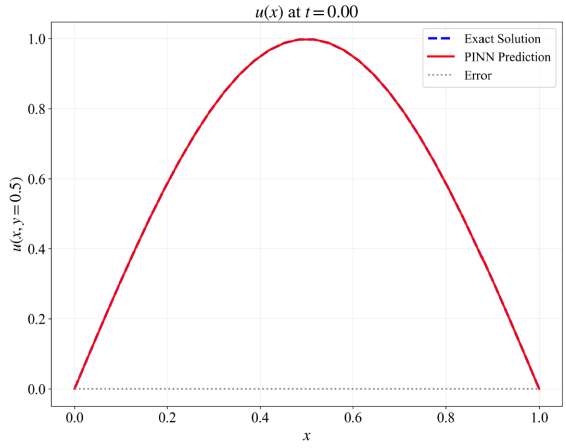


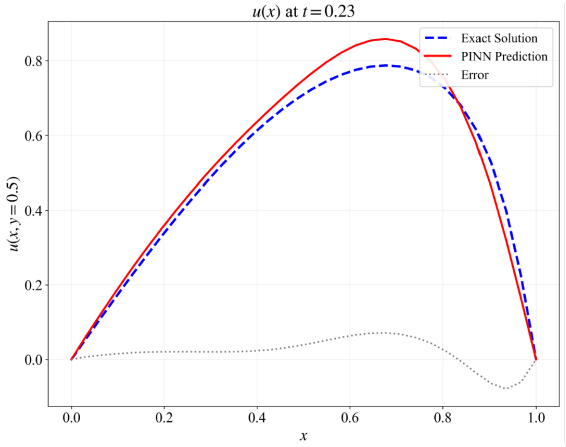
Figure 15. The loss convergence curve trained by the FNO-PINN method.

Figures 16 and 17 show the PINN results. The initial condition at $t = 0.00$ is captured well, but big deviations appear at later times. At $t = 0.23$, the PINN prediction has a clear shift in peak location

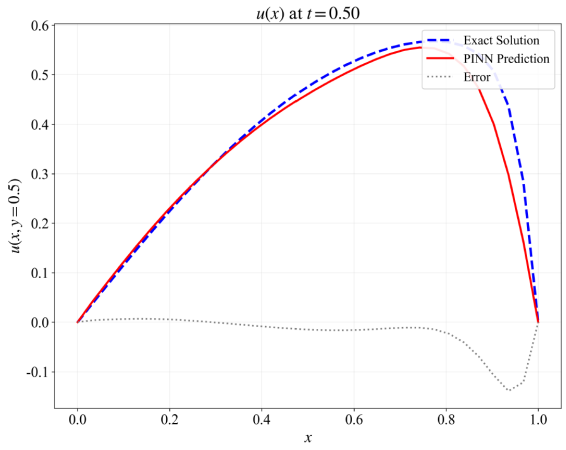
and amplitude. The error curve shows oscillations with maximum errors around 0.1. At $t = 0.50$, the overall shape is preserved, but differences remain near the boundaries.



(a) $t = 0.00$

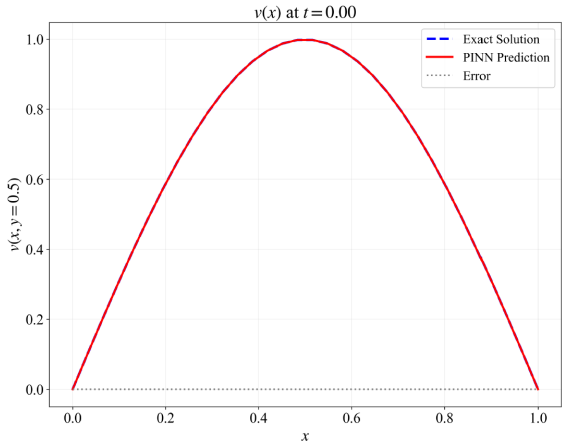


(b) $t = 0.23$

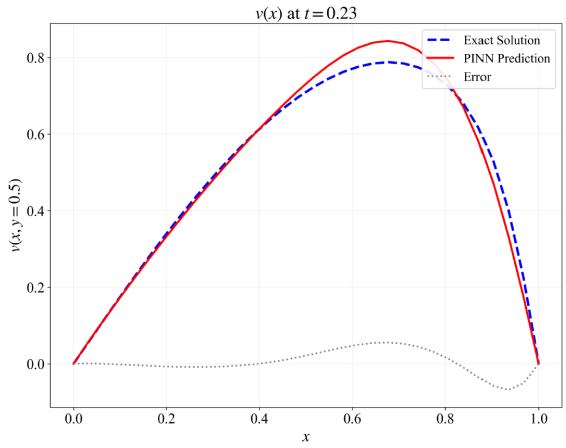


(c) $t = 0.50$

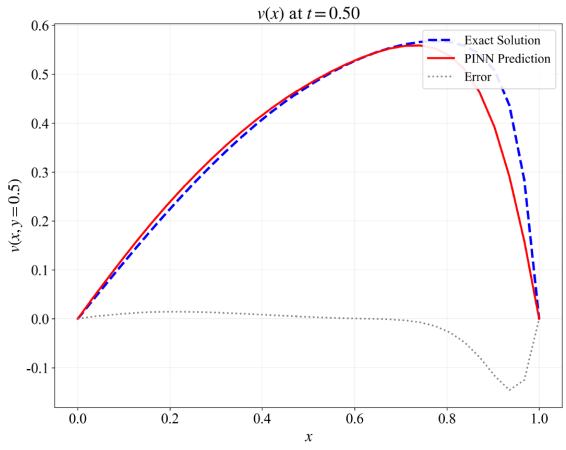
Figure 16. Comparison of the predicted value for variable u obtained by PINN and the exact solution.



(a) $t = 0.00$



(b) $t = 0.23$

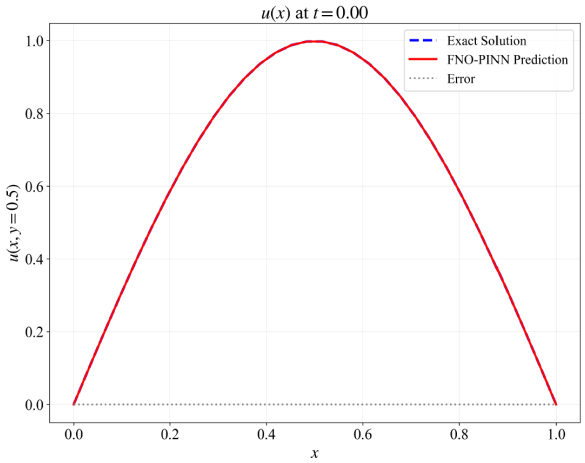


(c) $t = 0.50$

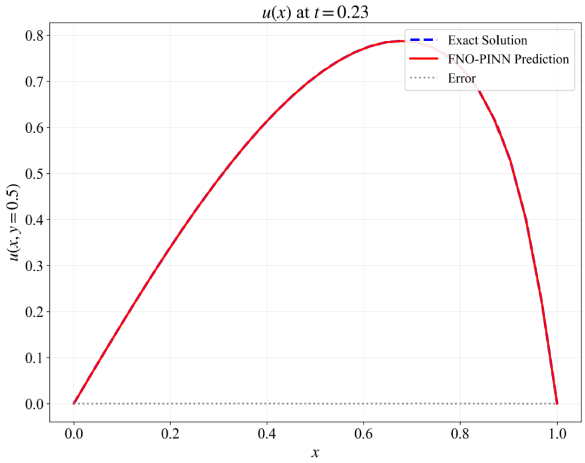
Figure 17. Comparison of the predicted value for variable v obtained by PINN and the exact solution.

Figures 18 and 19 show the FNO-PINN results for u and v at three times ($t = 0.00$, $t = 0.23$, and $t = 0.50$). The FNO-PINN predictions almost completely match the exact solution at all times. The

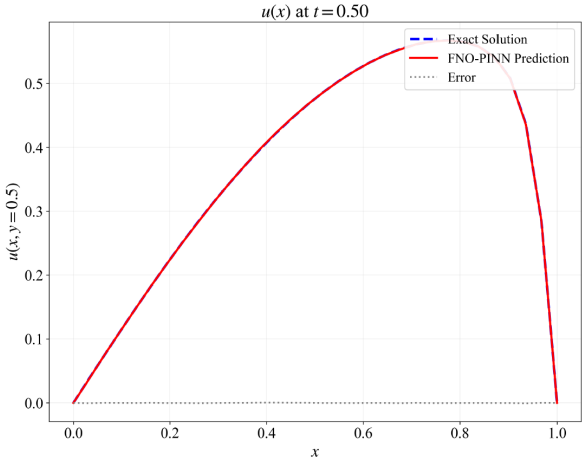
error stays near zero across the whole spatial domain.



(a) $t = 0.00$

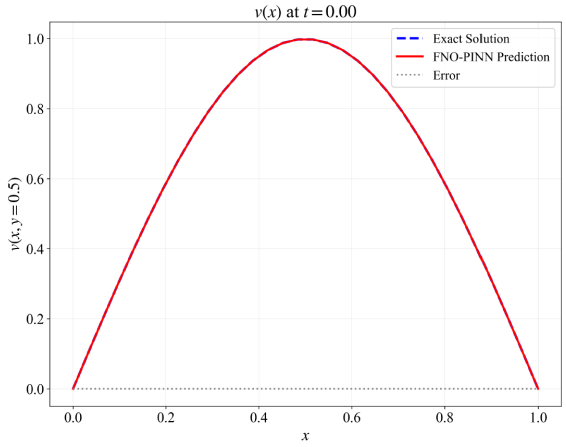


(b) $t = 0.23$

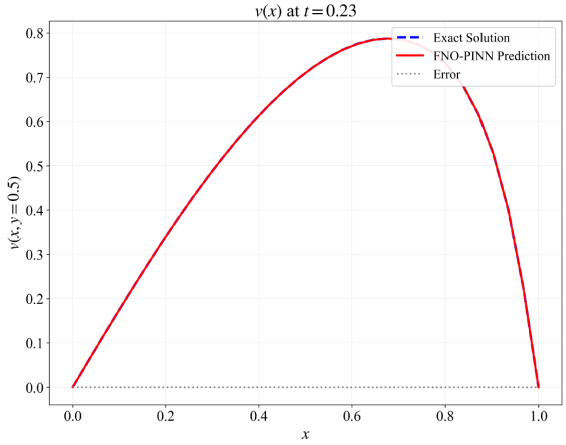


(c) $t = 0.50$

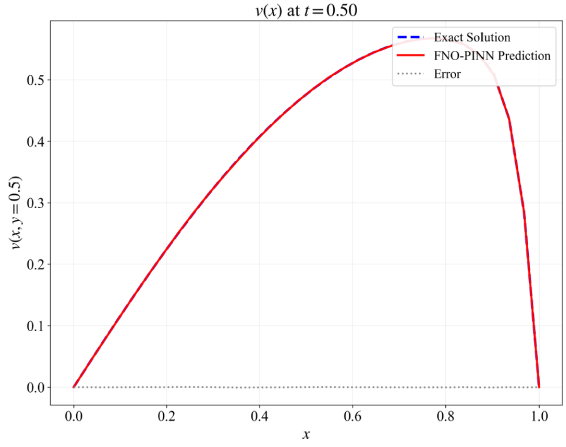
Figure 18. Comparison of the predicted value for variable u obtained by FNO-PINN and the exact solution.



(a) $t = 0.00$



(b) $t = 0.23$



(c) $t = 0.50$

Figure 19. Comparison of the predicted value for variable v obtained by FNO-PINN and the exact solution at different time steps.

The comparison confirms the same pattern as the previous examples: FNO-PINN's predictions are very close to the exact solution with small errors, while PINN's predictions have larger and more

scattered errors. This shows that PINN has trouble with nonlinear dynamics and steep gradients.

In conclusion, all three examples show that FNO-PINN outperform traditional PINN and MC methods. The FNO-PINN method achieves higher accuracy, more stable training, and faster convergence. These results confirm that FNO-PINN works well for PDEs with multiple-timescales and nonlinear dynamics.

5. Conclusions

This paper addresses the problem of solving PDEs and proposes a novel FNO-PINN method. Firstly, a hybrid method FNO-PINN combining the FNO and the PINN methods is proposed. The proposed method mainly consists of four core modules: enhanced FNO, high-precision differential calculation, multi-objective physical constraints, and a post-processing mechanism. By reasonably designing frequency domain transformation and convolution, spectral convolution layer, batch normalization and activation functions, combining finite differences to calculate partial derivatives with high accuracy, and introducing dynamic weight strategy loss function, the problem of solving the multi-timescale-coupled system in terms of accuracy and convergence speed is effectively addressed. Numerical experimental results show that the proposed FNO-PINN method is significantly superior to the traditional PINN method in terms of convergence speed and accuracy. The main innovation of this study lies in successfully combining the frequency domain learning ability of FNO with the physical constraint mechanism of PINN, and effectively solving the multi-timescale problem through multi-objective physical constraints and time-adaptive weight strategy loss function.

Several directions warrant further investigation. First, extending the framework to more complex fluid dynamics problems, such as incompressible Navier-Stokes equations, would demonstrate its broader applicability. This will require addressing additional challenges, including pressure-velocity coupling and the divergence-free constraint. Second, applying the framework to inverse problems for parameter identification is another promising direction. Third, testing on three-dimensional problems with complex geometries would further validate the scalability of the approach. Finally, it would also be valuable to include systematic comparisons with established numerical methods such as the finite element method (FEM) and the finite difference method (FDM) to provide a more complete evaluation of the proposed framework's performance.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This work was supported by grants from Ningxia higher education first-class discipline construction funding project (NXYLXK2017B09) and 2024 Graduate Innovation Project of North Minzu University NO.YCX24264.

Conflict of interest

The authors declare there are no conflicts of interest.

References

1. J. De Luca, Chemical principle and PDE of variational electrodynamics, *J. Differ. Equations*, **268** (2019), 272–300. https://doi.org/10.1016/j.jde.2019.08.020

2. J. Lv, J. Li, The delayed reaction-diffusion Filippov system with threshold control: Modeling and analysis of plateau pika-vegetation dynamics, *Commun. Nonlinear Sci. Numer. Simul.*, **152** (2026), 109366. https://doi.org/10.1016/j.cnsns.2025.109366

3. D. A. Nield, A. V. Kuznetsov, The Cheng-Minkowycz problem for natural convective boundary-layer flow in a porous medium saturated by a nanofluid, *Int. J. Heat Mass Transfer*, **52** (2009), 5792–5795. https://doi.org/10.1016/j.ijheatmasstransfer.2009.07.024

4. P. Suchde, J. Kuhnert, A meshfree generalized finite difference method for surface PDEs, *Comput. Math. Appl.*, **78** (2019), 2789–2805. https://doi.org/10.1016/j.camwa.2019.04.030

5. M. Li, C. S. Chen, Y. C. Hon, P. H. Wen, Finite integration method for solving multi-dimensional partial differential equations, *Appl. Math. Modell.*, **39** (2015), 4979–4994. https://doi.org/10.1016/j.apm.2015.03.049

6. Y. Liu, Y. Cheng, S. Chen, Y. Zhang, Krylov implicit integration factor discontinuous Galerkin methods on sparse grids for high dimensional reaction-diffusion equations, *J. Comput. Phys.*, **388** (2019), 90–102. https://doi.org/10.1016/j.jcp.2019.03.021

7. A. Townsend, S. Olver, The automatic solution of partial differential equations using a global spectral method, *J. Comput. Phys.*, **299** (2015), 106–123. https://doi.org/10.1016/j.jcp.2015.06.031

8. 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. https://doi.org/10.1016/j.jcp.2018.10.045

9. Y. Ren, H. Feng, S. Zhao, A FFT accelerated high order finite difference method for elliptic boundary value problems over irregular domains, *J. Comput. Phys.*, **448** (2022), 110762. https://doi.org/10.1016/j.jcp.2021.110762

10. J. Berg, K. Nyström, A unified deep artificial neural network approach to partial differential equations in complex geometries, *Neurocomputing*, **317** (2018), 28–41. https://doi.org/10.1016/j.neucom.2018.06.056

11. C. Beck, S. Becker, P. Grohs, N. Jaafari, A. Jentzen, Solving the Kolmogorov PDE by means of deep learning, *J. Sci. Comput.*, **88** (2021), 73. https://doi.org/10.1007/s10915-021-01590-0

12. L. Yuan, Y. Ni, X. Deng, S. Hao, A-PINN: Auxiliary physics informed neural networks for forward and inverse problems of nonlinear integro-differential equations, *J. Comput. Phys.*, **462** (2022), 111260. https://doi.org/10.1016/j.jcp.2022.111260

13. Y. Chen, L. Lu, G. E. Karniadakis, L. Dal Negro, Physics-informed neural networks for inverse problems in nano-optics and metamaterials, *Opt. Express*, **28** (2020), 11618–11633. https://doi.org/10.1364/OE.384875

14. G. Pang, L. Lu, G. E. Karniadakis, fPINNs: Fractional physics-informed neural networks, *SIAM J. Sci. Comput.*, **41** (2019), 2603–2626. https://doi.org/10.1137/18M1229845

15. S. Wang, H. Wang, P. Perdikaris, On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks, *Comput. Methods Appl. Mech. Eng.*, **384** (2021), 113938. https://doi.org/10.1016/j.cma.2021.113938

16. S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, preprint, arXiv:2001.04536.

17. J. Li, J. Chen, B. Li, Gradient-optimized physics-informed neural networks (GOPINNs): A deep learning method for solving the complex modified KDV equation, *Nonlinear Dyn.*, **107** (2022), 781–792. https://doi.org/10.1007/s11071-021-06996-x

18. A. D. Jagtap, G. E. Karniadakis, Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, *Commun. Comput. Phys.*, **28** (2020), 2002-2041. https://doi.org/10.4208/cicp.OA-2020-0164

19. L. Guo, H. Wu, X. Yu, T. Zhou, Monte Carlo fPINNs: Deep learning method for forward and inverse problems involving high dimensional fractional partial differential equations, *Comput. Methods Appl. Mech. Eng.*, **400** (2022), 115523. https://doi.org/10.1016/j.cma.2022.115523

20. Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, et al., Fourier neural operator for parametric partial differential equations, preprint, arXiv:2010.08895.

21. Z. K. Lawal, H. Yassin, D. T. C. Lai, A. C. Idris, Modeling the complex spatio-temporal dynamics of ocean wave parameters: A hybrid PINN-LSTM approach for accurate wave forecasting, *Measurement*, **252** (2025), 117383. https://doi.org/10.1016/j.measurement.2025.117383

22. S. Cai, Z. Wang, S. Wang, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks for heat transfer problems, *ASME. J. Heat Mass Transfer.*, **143** (2021), 060801. https://doi.org/10.1115/1.4050542

23. B. Bahtiri, B. Arash, S. Scheffler, M. Jux, R. Rolfes, A thermodynamically consistent physics-informed deep learning material model for short fiber/polymer nanocomposites, *Comput. Methods Appl. Mech. Eng.*, **427** (2024), 117038. https://doi.org/10.1016/j.cma.2024.117038

24. J. Zhao, Highly accurate compact mixed methods for two point boundary value problems, *Appl. Math. Comput.*, **188** (2007), 1402–1418. https://doi.org/10.1016/j.amc.2006.11.006

25. C. Shu, S. Osher, Efficient implementation of essentially non-oscillatory shock-capturing schemes, *J. Comput. Phys.*, **77** (1988), 439–471. https://doi.org/10.1016/0021-9991(88)90177-5

AIMS Press