Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2025-0185

Vol. **40**, No. 1, pp. 61-87
July 2026

# Structured First-Layer Initialization Pre-Training Techniques to Accelerate Training Process Based on $\varepsilon$-Rank

Tao Tang[4,7], Jiang Yang[1,3,5,6,*], Yuxiang Zhao[1] and Quanhui Zhu[1,2]

[1] *Department of Mathematics, Southern University of Science and Technology, Shenzhen, China.*
[2] *Department of Mathematics, National University of Singapore, Singapore.*
[3] *SUSTech International Center for Mathematics, Shenzhen, China.*
[4] *School of Mathematics and Statistics, Guangzhou Nanfang College, Guangzhou, China.*
[5] *Guangdong Provincial Key Laboratory of Computational Science and Material Design, Southern University of Science and Technology, Shenzhen, China.*
[6] *National Center for Applied Mathematics Shenzhen (NCAMS), Shenzhen, 518055, P.R. China.*
[7] *Zhuhai SimArk Technology Co., LTD, Zhuhai, Guangdong, China.*

Received 17 July 2025; Accepted (in revised version) 18 November 2025

**Abstract.** Training deep neural networks for scientific computing remains computationally expensive due to the slow formation of diverse feature representations in early training stages. Recent studies [37] identify a staircase phenomenon in training dynamics, where loss decreases are closely correlated with increases in $\varepsilon$-rank, reflecting the effective number of linearly independent neuron functions. Motivated by this observation, this work proposes a structured first-layer initialization (SFLI) pre-training technique to enhance the diversity of neural features at initialization by constructing $\varepsilon$-linearly independent neurons in the input layer. We present systematic initialization schemes compatible with various activation functions and integrate the strategy into multiple neural architectures, including modified multi-layer perceptrons and physics-informed residual adaptive networks. Only needing to add one line of code to conventional stochastic gradient descent algorithms, extensive numerical experiments on function approximation and PDE benchmarks, demonstrate that SFLI significantly improves the initial $\varepsilon$-rank, accelerates convergence, mitigates spectral bias, and enhances prediction accuracy.

**AMS subject classifications**: 68T07, 65Z05, 35Q68

**Key words**: Staircase phenomenon, $\varepsilon$-rank, structured first-layer initialization, deep neural network, training acceleration.

*Corresponding author. *Email addresses:* ttang@nfu.edu.cn (T. Tang), yangj7@sustech.edu.cn (J. Yang), 12131241@mail.sustech.edu.cn (Y. Zhao), zhuqh@nus.edu.sg (Q. Zhu)

 ©2026 Global-Science Press

# 1 Introduction

Neural networks have become a cornerstone of modern machine learning, achieving remarkable accuracy in both classical machine learning tasks and emerging areas such as scientific computing and partial differential equation (PDE) modeling. Despite their success, training such models remains computationally intensive, often requiring large-scale resources and prolonged optimization. This has motivated a broad range of efforts to accelerate the training process, including architectural innovations, advanced optimization techniques, and improvements in loss function design.

Among these efforts, particular attention has been paid to the learning dynamics of neural networks [6, 10, 17, 18, 22, 40]. A key observation in training dynamics is the so-called frequency principle (F-Principle) or spectral bias [24, 34–36, 39], which states that neural networks tend to fit low-frequency components of the target function earlier in training. To address this, multiscale network designs like MscaleDNN [19, 20] have been proposed, which incorporate hierarchical frequency encodings. The performance of such architectures strongly depends on the behavior of the first hidden layer, particularly the diversity of its initial activations, a critical aspect that remains insufficiently studied.

A standard multi-layer perceptron (MLP) can be formulated as follows:

$$\begin{cases} y_0 = x, \\ y_l = H(y_{l-1};\theta_l), \quad l = 1,\cdots,L, \\ y = \beta \cdot y_L, \end{cases} \tag{1.1}$$

where $x \in \mathbb{R}^d$ is the input, $y_l \in \mathbb{R}^{n_l}$ is the neurons of the $l$-th hidden layer, and $\beta \in \mathbb{R}^{n_L}$ is the coefficients in the output layer. Each layer mapping $H$ represents the structure of the hidden layer, and a fully connected layer can be explicitly given by $H(y_{l-1};\theta_l) = \sigma(W_l y_{l-1} + b_l)$, where $W_l \in \mathbb{R}^{n_{l+1}\times n_l}$, $b_l \in \mathbb{R}^{n_{l+1}}$, are trainable parameters. The activation function $\sigma$ is applied element-wise. The neurons $y_L(x)$ in the final hidden layer can be viewed as a collection of scalar neuron functions defined on the input domain, and the final output $y$ of the network is a linear combination of these functions. This perspective aligns with interpretations in the deep finite element method [32] and the finite neuron method [33], where $y_L(x)$ serves as a set of basis functions.

The training dynamics of deep neural networks have attracted significant interest, particularly in understanding how low-dimensional representations are formed during optimization [1, 23]. Recent work [37] has identified *staircase phenomenon*: **In training dynamics, the loss function often decreases rapidly along with a significant growth of linear independence of neuron functions.** A consistently low $\varepsilon$-rank of $y_L$ during training indicates a lack of functional diversity among neurons, which in turn limits the expressive power of the network. Such limitations pose a critical challenge for problems in scientific computing, where resolving fine-scale or high-frequency structures is essential.

Motivated by these insights, this work introduces a novel pre-training strategy

termed structured first-layer initialization (SFLI), designed to increase the $\varepsilon$-rank of neuron functions at initialization. By enforcing $\varepsilon$-linear independence through carefully constructed weights in the first hidden layer, the method enhances feature diversity and accelerates convergence, while preserving numerical stability and incurring no additional computational cost.

The main contributions of this work are summarized as follows:

1. We propose a novel structured first-layer initialization pre-training strategy, which enhances initial feature diversity of neural functions and accelerates the training process. The method is activation-function agnostic, compatible with a wide range of neural architectures, and incurs negligible computational overhead, making it broadly applicable and easy to integrate into existing frameworks.

2. We demonstrate the effectiveness of structured first-layer initialization across various function approximation and PDE-solving benchmarks, where the method consistently improves prediction accuracy, convergence speed, and numerical stability under different training scenarios, while also mitigating spectral bias through improved feature diversity.

The remainder of the paper is organized as follows. Section 2 reviews the concept of $\varepsilon$-rank and introduces the staircase phenomenon. Section 3 presents the proposed structured first-layer initialization and discusses its implementation across various network architectures. Section 4 provides numerical results to illustrate the effectiveness of the method. Conclusions and future works are summarized in Section 5.

## 2 Staircase phenomenon

In this section, we will first present a short recall on the $\varepsilon$-rank, which is introduced to quantitatively measure the linear independence of neuron functions, as formally defined below:

**Definition 2.1** ( [37]). Let $u(x;\theta)$ denote a neural network with neuron functions $\{\phi_j\}_{j=1}^n$ in its final hidden layer. Define the Gram matrix $(M_u)_{ij}=\int_\Omega \phi_i(x;\theta)\phi_j(x;\theta)\mathrm{d}x$. Then the $\varepsilon$-rank of $M_u$ is given by

$$r_\varepsilon(M_u)=\#\{\lambda\in\lambda(M_u)\,|\,\lambda>\varepsilon\}, \tag{2.1}$$

representing the number of eigenvalues $\lambda(M_u)$ exceeding a threshold $\varepsilon$.

Tracking $\varepsilon$-rank reveals the staircase phenomenon in training dynamics (Fig. 1): the loss function $\mathcal{L}(u_n)$ decreases sharply when $r_\varepsilon(M_u)$ increases significantly. This behavior has been observed consistently across different tasks and architectures in [37].

This phenomenon is theoretically justified by the following lower bound on the loss:

$$\sqrt{\mathcal{L}(u_n)}\geq\frac{1}{C_s}\left(\sqrt{\mathrm{dist}(u^*,\mathcal{F}_p)}-C(p+1)(n-p)^2\varepsilon\right), \tag{2.2}$$

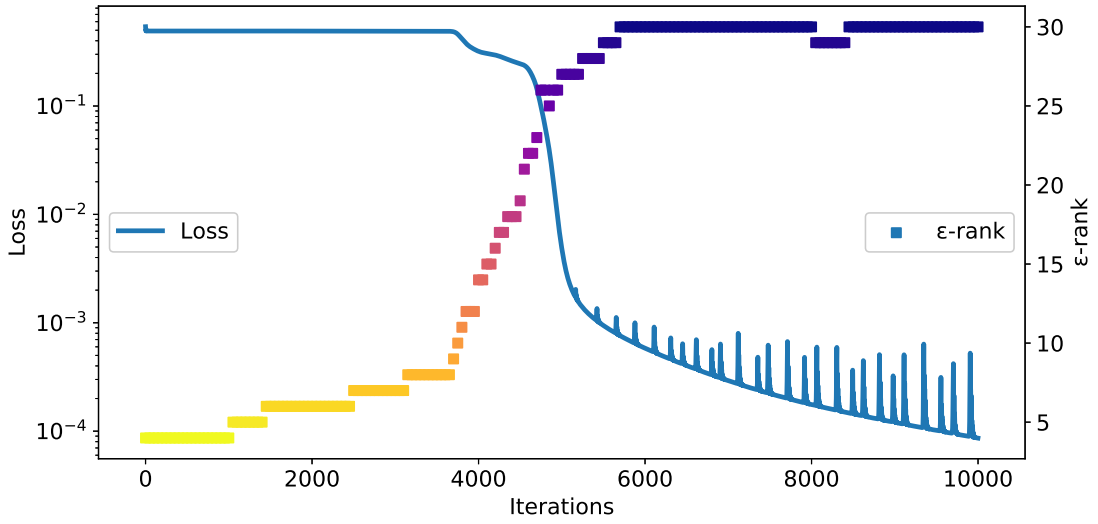


Figure 1: Staircase phenomenon in a regression task with target function $f(x)=\sin 20x$.

where $u_n \in \mathcal{F}_n$ is a neural network of width $n$, $p$ is the $\varepsilon$-rank of $u_n$, i.e., $r_\varepsilon(M_{u_n})=p$, and $\mathcal{F}_p$ denotes the function class with $p$ neurons in the last hidden layer. This inequality shows that meaningful reduction in loss requires an increase in the $\varepsilon$-rank.

Taken together, these observations suggest that a sufficiently large $\varepsilon$-rank is a prerequisite for significant loss decay. Empirical evidence in [37] further supports this claim: training curves often exhibit prolonged plateaus when $r_\varepsilon(M_u)$ remains stagnant, followed by decreases only after $\varepsilon$-rank jumps.

Standard initialization schemes, such as Xavier initialization [8], typically result in a low initial $\varepsilon$-rank, thereby requiring extended training to overcome the rank bottleneck. Fig. 2 demonstrates the observation that within the same deep neural network, the $\varepsilon$-rank of the subsequent layer is higher than that of the previous layer. These insights have inspired pre-training strategies that directly construct $\varepsilon$-linearly independent neuron functions in the first hidden layer using deterministic weight initialization [37]. This approach effectively elevates the initial $\varepsilon$-rank to $n$, bypassing the early-stage stagnation and accelerating convergence in practice.

This unified perspective, linking $\varepsilon$-rank to training dynamics, provides actionable guidelines for improving neural network performance. In the next section, we build on this foundation to propose a general, structured first-layer initialization strategy that systematically improves the initial representational capacity, with a focus on accelerating the training dynamics.

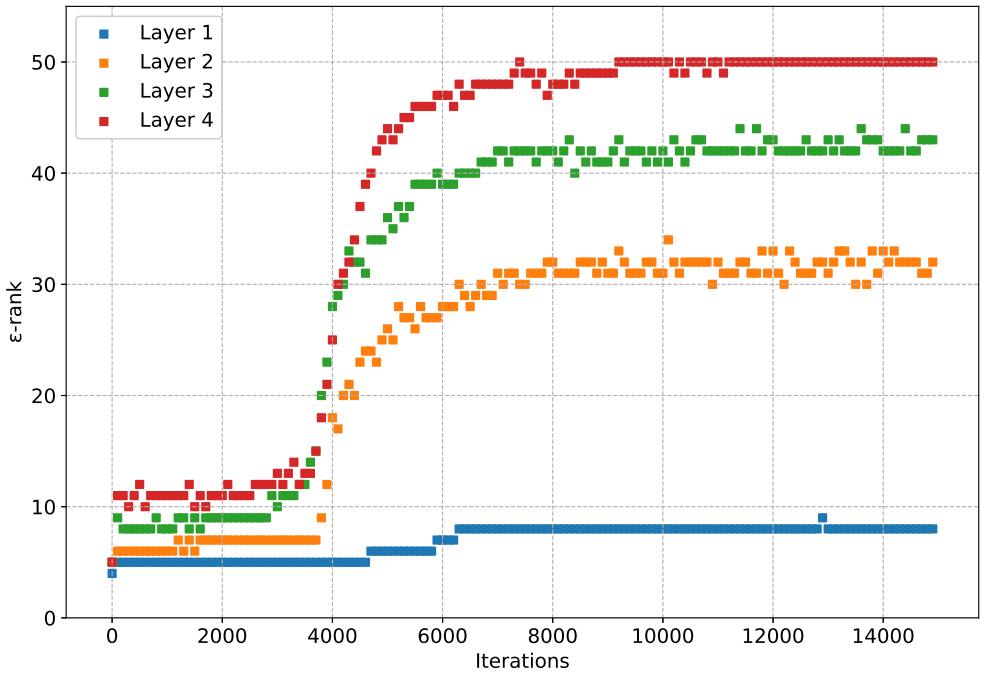


Figure 2: Layer-wise staircase phenomenon within the same neural network. The width is fixed $n_1=n_2=n_3=n_4=50$.

# 3 Structured first-layer initialization

## 3.1 Pre-activation method

Following the findings in [37], the first hidden layer plays a critical role in learning meaningful feature representations. Consider the neurons in the first hidden layer defined by:

$$F(x)=\sigma(Wx+b), \tag{3.1}$$

where $x\in\Omega\subset\mathbb{R}^d$ is the input, $b=[b_1,\cdots,b_n]^\top\in\mathbb{R}^n$, $W=[w_1,\cdots,w_n]^\top\in\mathbb{R}^{n\times d}$ are trainable parameters, and $\sigma$ is an element-wise activation function. The vector-valued function $F(x)\in\mathbb{R}^n$ represents the output of the first hidden layer, where $F(x):=[F_1(x),\cdots,F_n(x)]^\top$ and each $F_i(x)$ is a scalar neuron function.

The objective of structured first-layer initialization is to construct a set of neuron functions that are approximately $\varepsilon$-linearly independent at initialization. This is achieved by strategically designing the weights and biases in the first hidden layer so that the neural functions $\{F_i(x)\}$ are well-separated in the input domain. The motivation stems from classical numerical methods such as finite element schemes, where basis functions are spatially localized and evenly distributed.

Specifically, we reparameterize each neuron function in the first hidden layer as:

$$F_i(x;w_i,b_i)=\sigma(w_i\cdot x+b_i)=\sigma(\gamma(\alpha_i\cdot x+c_i)),\quad i=1,\cdots,n, \tag{3.2}$$

where $w_i=\gamma\alpha_i$, $b_i=\gamma c_i$. $c_i\geq 0$ is the shift distributed uniformly in an interval with respect to $\Omega$, and $\alpha_i$ determines the orientation of the hyperplane. The parameter $\gamma>0$ controls the localization of the neural function, with a recommended practical choice:

$$\gamma=C\cdot\frac{n^{1/d}-1}{|\Omega|^{1/d}},\quad C\in[0.5,2]. \tag{3.3}$$

For $d=1$, this choice is consistent with piecewise linear basis functions in the finite element method, where $\gamma=\frac{n-1}{b-a}$. A similar scaling parameter has also been used in batch normalization techniques [12] and adaptive activation functions [14]. However, in these methods, the scaling factor is a trainable quantity that dynamically changes during training and affects multiple layers, whereas in SFLI it is a fixed initialization hyperparameter applied only to the first layer to enhance the initial feature diversity.

The overall structure of SFLI is illustrated in Fig. 3. To provide an intuitive understanding, Fig. 4 visualizes the output behaviors of different activation functions under SFLI with $n=5$ neurons. Below, we list several typical activation functions along with their associated weight initialization and structural characteristics used in the SFLI method:

- **Cosine:** $\sigma(x)=\cos(x)$. The weight vectors $\alpha_i$ are sampled independently from the standard normal distribution, i.e., $\alpha_i\sim\mathcal{N}(\mathbf{0},\boldsymbol{I}_d)$.

- **Hyperbolic Tangent (Tanh):** $\sigma(x)=\tanh(x)$. The weights $\alpha_i$ are uniformly sampled from the unit sphere, i.e., $\alpha_i\in\mathbb{S}^d,\|\alpha_i\|_2=1$, to ensure directional diversity in the neuron responses.

- **Hat:** $\sigma(x)=\max(0,1-|x|)$. Similar to Tanh, the weight vectors are chosen from the unit sphere: $\alpha_i\in\mathbb{S}^d$. This ensures spatial coverage of the support regions of hat functions over the domain.

- **Gaussian:** $\sigma(x)=e^{-|x|^2}$. Unlike the previous activation functions, which are based on linear transformations $w_i\cdot x+b_i$, the Gaussian activation adopts a radial structure centered at specific locations in the input domain. Specifically, each neuron in the first hidden layer takes the form:

  $$F_i(x;\gamma,c_i)=\exp\left(-\gamma^2\|x-c_i\|^2\right),\quad i=1,\cdots,n,$$

  where $\gamma>0$ is a shape parameter that controls the sharpness of each Gaussian bump, and $c_i\in\mathbb{R}^d$ denotes the center of the $i$-th neuron uniformly distributed in $\Omega$.

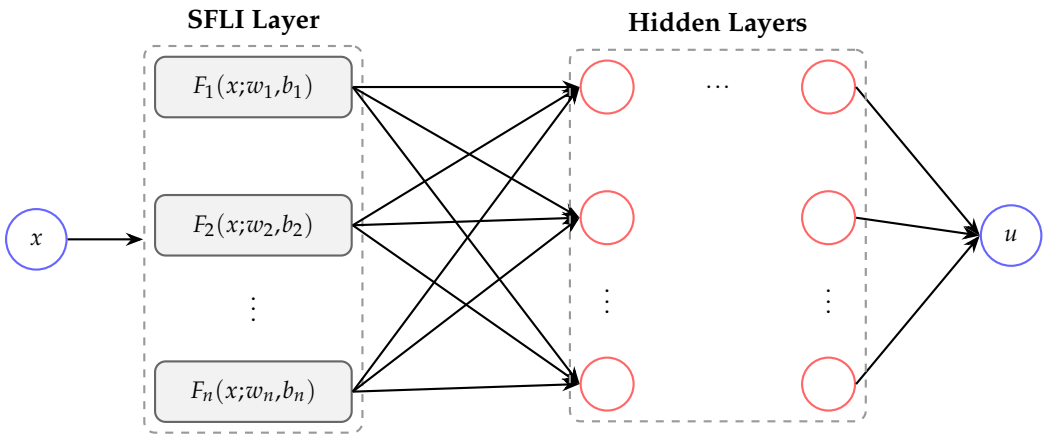


Figure 3: Architecture of the neural network with Structured First-Layer Initialization (SFLI).

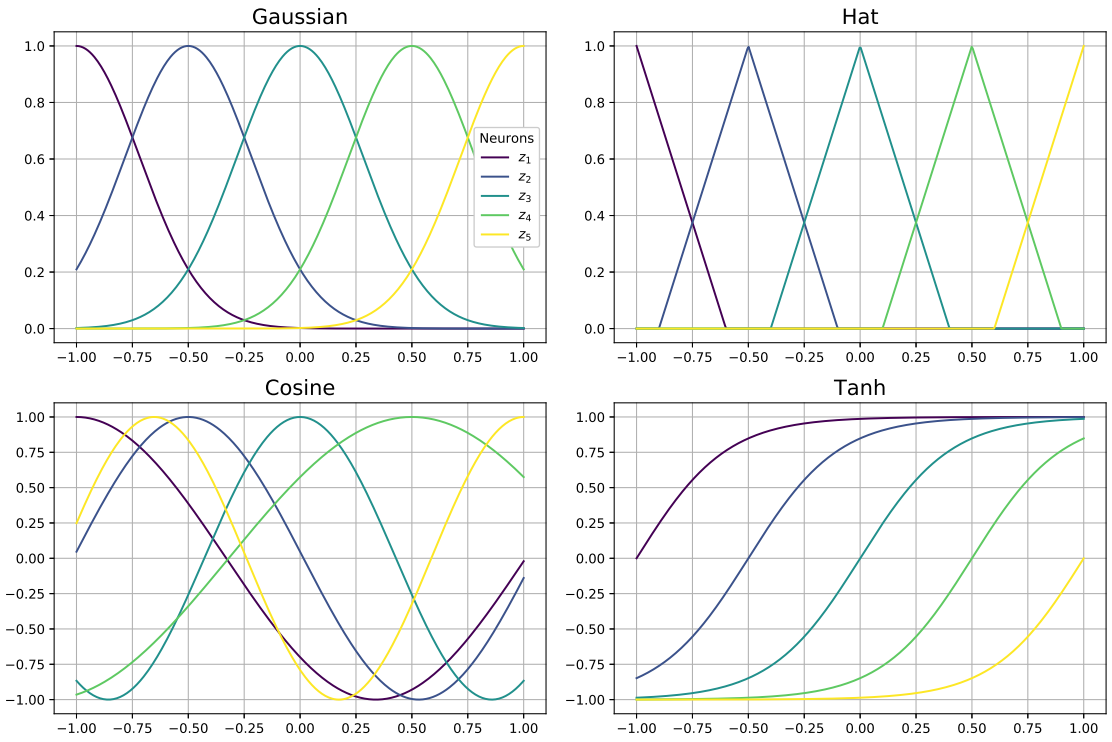


Figure 4: Visualization of the first-layer outputs using different activation functions under SFLI with $n=5$ neurons.

Importantly, SFLI modifies only the initialization of the first layer and does not change optimization, i.e., optimize $w_i$,$b_i$ directly rather than the decomposed parameters $\alpha_i$,$c_i$ and $\gamma$. The activation function and parameter settings are applied exclusively at initialization and do not affect subsequent layers, ensuring compatibility with standard training pipelines and incurring negligible computational cost.

The goals of the SFLI pre-training strategy can be summarized as follows:

- Promote the diversity of the neurons over the input domain $\Omega$.
- Ensure that the $\varepsilon$-rank of the initial feature set $\{F_i\}$ approaches $n$ with high probability.
- Maintain full compatibility with standard architectures without introducing additional training cost.

These principles underpin the design of the SFLI strategy and are further validated through empirical studies in the next section.

### 3.2 Integrating with various neural network architectures

The proposed structured first-layer initialization strategy can be seamlessly integrated into a wide range of neural network architectures, including Residual Neural Network (ResNet) [11], modified MLP [30], and Physics-Informed Residual AdapTivE Network (PirateNet) [27]. These models are representative of architectures commonly used in scientific computing and physics-informed learning.

The modified MLP enhances the standard architecture by introducing dual encoders and adaptive gating, which improves convergence and residual minimization in PDE applications. PirateNet extends this by incorporating adaptive residual blocks, allowing for deeper networks with stable training dynamics.

PirateNet incorporates adaptive residual connections to mitigate the issue of pathological initialization, enabling stable and efficient training of physics-informed neural networks with deeper architectures.

As representative cases, we describe the network structures used in our numerical experiments. First, we introduce the modified MLP architecture:

$$U=\sigma(W^{(u)}x+b^{(u)}),\quad V=\sigma(W^{(v)}x+b^{(v)}), \tag{3.4}$$

followed by $L$ hidden layers:

$$\begin{aligned} &y_0=x,\\ &z_l=\sigma(W_l y_{l-1}+b_l),\\ &y_l=z_l\odot U+(1-z_l)\odot V,\quad l=1,\cdots,L. \end{aligned} \tag{3.5}$$

The final network output is also linear combinations of the last hidden layer

$$y=\beta\cdot y_L.$$

Here, $\odot$ denotes an element-wise multiplication, and

$$\theta=\{W^{(u)},b^{(u)},W^{(v)},b^{(v)},(W_l,b_l)_{l=1}^{L},\beta\}$$

are all trainable parameters.

PirateNet builds on this structure by adding $L$ residual blocks. The architecture starts with the same encoders and first-layer MLP as above. For $l=1,\cdots,L$, the residual block $y_{l+1}=H(y_l;\theta_l,U,V)$ is defined as:

$$\begin{aligned}
&f_l=\sigma(W_l^{(1)}y_l+b_l^{(1)}),\\
&z_l^{(1)}=f_l\odot U+(1-f_l)\odot V,\\
&g_l=\sigma(W_l^{(2)}z_l^{(1)}+b_l^{(2)}),\\
&z_l^{(2)}=g_l\odot U+(1-g_l)\odot V,\\
&h_l=\sigma(W_l^{(3)}z_l^{(2)}+b_l^{(3)}),\\
&y_{l+1}=\alpha_l h_l+(1-\alpha_l)y_l,
\end{aligned}\tag{3.6}$$

and the full forward pass is:

$$\begin{cases}
U=\sigma(W^{(u)}x+b^{(u)})\\
V=\sigma(W^{(v)}x+b^{(v)}),\\
z=\sigma(W_1x+b_1),\\
y_1=z\odot U+(1-z)\odot V,\\
y_{l+1}=H(y_l;\theta_l,U,V),\quad l=1,\cdots,L,\\
y=\beta\cdot y_L.
\end{cases}\tag{3.7}$$

The trainable parameters include all encoder and block components:

$$\theta=\left\{W^{(u)},b^{(u)},W^{(v)},b^{(v)},(W_1,b_1),(W_l^{(1,2,3)},b_l^{(1,2,3)},\alpha_l)_{l=1}^{L},\beta\right\}.$$

Each residual block contains three fully connected layers with gating and an adaptive residual connection. Initializing $\alpha_l=0$ reduces the network to a shallow modified MLP, improving trainability in early stages. Expressivity gradually increases as training progresses.

To promote a high initial $\varepsilon$-rank, we apply SFLI to the encoders $U$ and $V$ defined in (3.4). Numerical evidence shows that this improves both convergence rate and final accuracy. Fig. 5 presents a schematic diagram of PirateNet architecture with SFLI applied to its encoder layer.

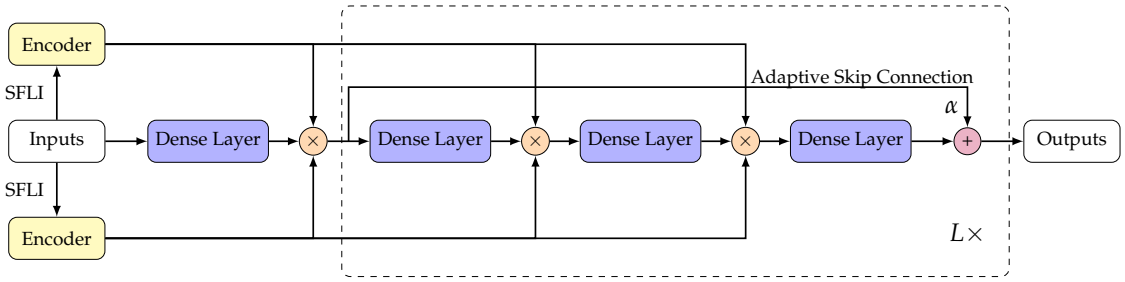


Figure 5: Architecture of PirateNet with Structured First-Layer Initialization (SFLI).

# 4 Numerical experiments

This section presents a series of numerical experiments designed to evaluate the effectiveness of the proposed structured first-layer initialization (SFLI) strategy in two contexts: function approximation and the solution of PDEs using physics-informed neural networks (PINNs). We focus on evaluating training efficiency, approximation accuracy, and the evolution of the $\varepsilon$-rank.

Unless otherwise mentioned, the default training setup across all experiments is as follows. The networks are trained using the Adam optimizer with an initial learning rate of $10^{-3}$. For one dimensional problems, the dataset is generated in the uniform grid points, while for higher dimensional problems, the training dataset is uniformly resampled from the input domain every 10 iterations. For function approximation tasks, each model adopts a fully connected architecture with 3 hidden layers. The hyperbolic tangent function is adopted as the default activation function for the baseline models. Xavier initialization is applied as default initialization for all layers. The threshold for computing the $\varepsilon$-rank is set to $10^{-6}$. More detailed experiment configurations and hyperparameter settings of each example are provided in Appendix A. The implementation and all experiment codes are publicly available at our GitHub repository `https://github.com/zyx979/SFLI`.

## 4.1 Function approximations

As discussed in [37], using a large initialization variance can lead to a higher initial $\varepsilon$-rank. In this example, we show that the proposed SFLI method performs better than a large variance initialization.

**Example 4.1** (Comparison with Large Variance Initialization)**.** Consider the target function

$$f(x)=\cos x+\cos 2x+\cos 30x, \quad x\in[-1,1]. \tag{4.1}$$

We compare the performance of the proposed SFLI method against a large variance initialization strategy in approximating this function. For a fair comparison, the variance of the weights for the large-variance initialization is set to $\sigma^2=\gamma^2/3$, which is significantly

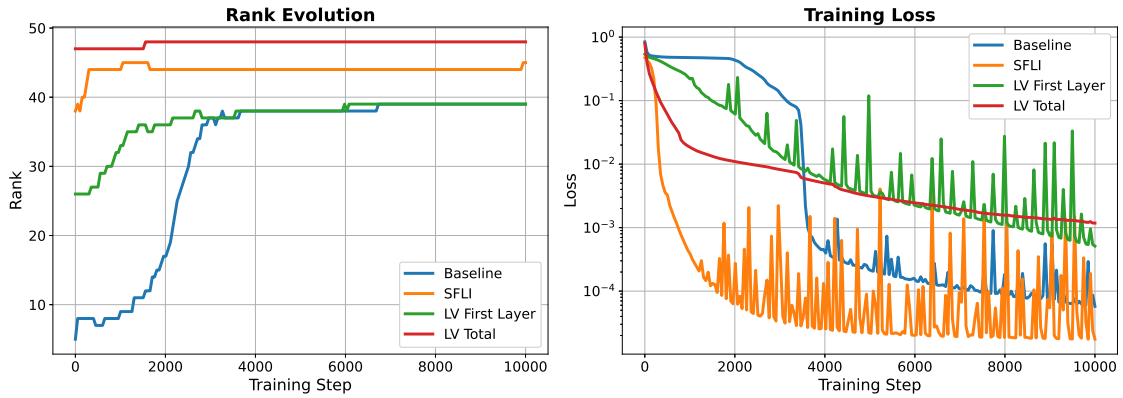


Figure 6: Comparison between SFLI and large variance (LV) initialization on Example 4.1. Left: $\varepsilon$-rank dynamics; Right: Training loss curves. The detailed hyper-parameter settings are presented in Table 3.

larger than the standard Xavier initialization variance of $\sigma^2=1/n$ for a layer with $n$ neurons. The test configurations include: (i) default Xavier initialization (Baseline), (ii) large variance initialization on first layer (LV First Layer), (iii) large variance initialization on all layers (LV Total), and (iv) SFLI method. Activation function is set to tanh for all layers.

The results are presented in Fig. 6. The left panel shows the evolution of the $\varepsilon$-rank during training, while the right panel displays the corresponding training loss curves. Notably, the large variance initialization on all layers produces a slightly higher initial $\varepsilon$-rank than SFLI, but its convergence slows substantially after the initial rapid decline in loss. In contrast, SFLI exhibits a steady and monotonic reduction in training loss throughout the optimization process. This suggests that, although increasing the initialization variance can improve the initial $\varepsilon$-rank and early-stage accuracy, it reduces training efficiency at later stages. By introducing structured diversity at initialization, SFLI preserves training stability and achieves superior overall performance.

**Example 4.2** (High- and Low-Frequency 2D Function)**.** Consider the following composite target function that contains both high- and low-frequency components:

$$f(\mathbf{x})=\cos(x_1)\cos(x_2)+\cos(10x_1)\cos(10x_2),\quad \mathbf{x}\in[-1,1]^2.$$

This benchmark function is designed to test the effectiveness of the SFLI pre-training strategy under varying frequency conditions. In this example, we assess the impact of SFLI with different activation functions. The first layer uses various activation functions, including Gaussian, Tanh, Cosine, and Hat activations. We compare the networks initialized with SFLI against the baseline counterparts (without SFLI) to evaluate their effects on $\varepsilon$-rank and convergence.

As shown in Fig. 7, all SFLI-based methods demonstrate superior performance compared to the baseline. Specifically, the SFLI variants exhibit high initial $\varepsilon$-ranks followed

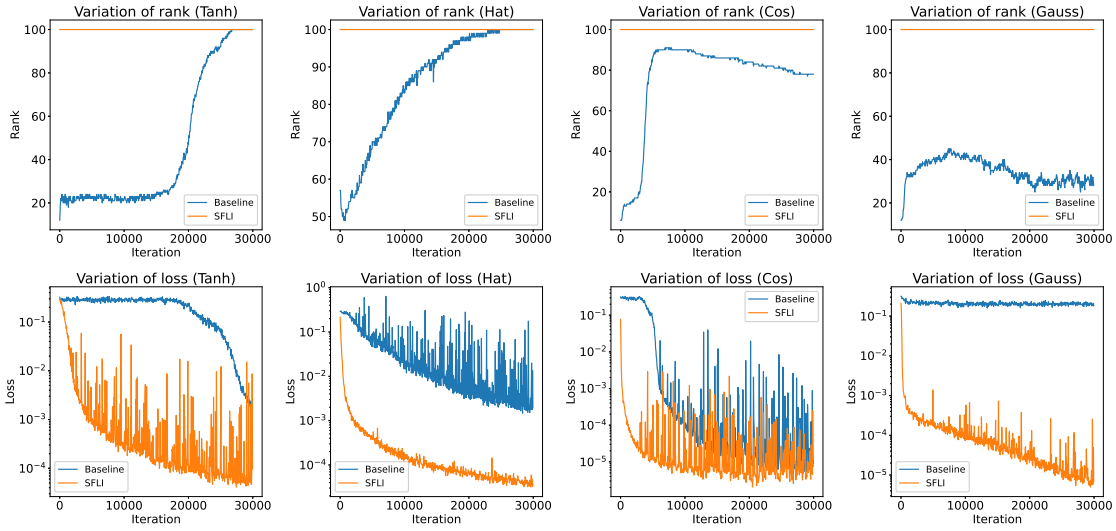


Figure 7: Comparison of $\varepsilon$-rank dynamics (top) and training loss (bottom) across different activation functions on Example 4.2. The plots compare the baseline (no SFLI) vs. SFLI with Gaussian, Tanh, Cosine, and Hat activations. The detailed hyper-parameter settings are presented in Table 3.

by rapid loss reduction during training, while the baseline remains trapped in a rank plateau with sluggish optimization progress except for the case with the cosine activation, where the spectral structure of the target function naturally aligns with the activation pattern. Nevertheless, SFLI still achieves faster convergence in this case. These empirical results demonstrate that the SFLI pre-training strategy effectively accelerates convergence and enhances accuracy through $\varepsilon$-rank.

**Example 4.3** (Discontinuous and Multiscale 1D Function)**.** Consider the following piecewise-defined target function:

$$f(x)=\begin{cases}(x^2+1)\sin(80x), & \text{if } -1\le x<-\frac{1}{3},\\ (-2x+3)\cos(10x), & \text{if } -\frac{1}{3}\le x<\frac{1}{3},\\ x^3-x, & \text{if } \frac{1}{3}\le x\le 1.\end{cases}$$

This function exhibits both discontinuities and multi-scale characteristics, making it a suitable test case for evaluating the spectral bias in neural network training. The Fourier series representation of $f$ is given by:

$$f(x):=\sum_{k=-\infty}^{\infty}\hat{f}_k e^{ik\pi x}.$$

To quantify the spectral bias, we compute the spectral error of a neural network predic-

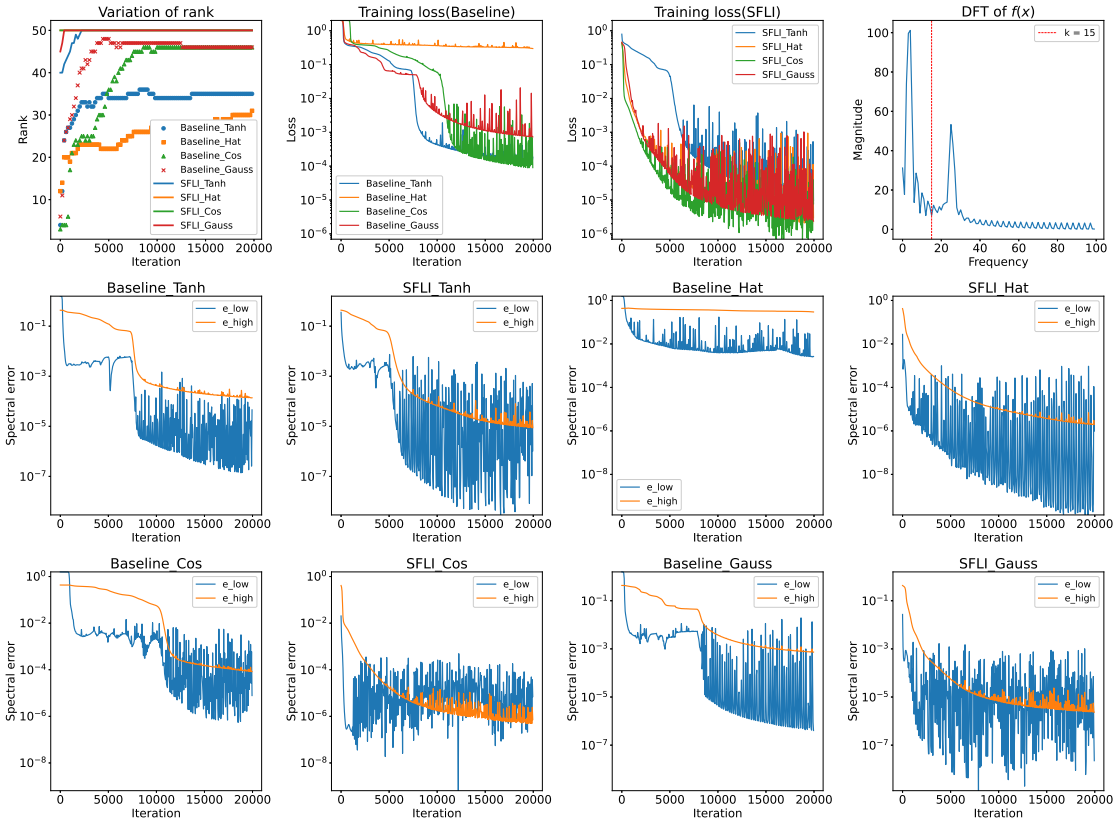


Figure 8: Spectral analysis on Example 4.3 using different SFLI strategies. The target function contains both low- and high-frequency components. The top row displays (from left to right): (1) the evolution of $\varepsilon$-rank during training; (2) training loss curves without SFLI; (3) training loss curves with SFLI; (4) spectrum of the target function, dividing low-frequency and high-frequency components by $k=15$; The next two rows show the spectral fitting errors for low- and high-frequency components with respect to four activation functions (Tanh, Hat, Cos, Gauss), both with and without SFLI. Here, $e_{\text{low}}$ and $e_{\text{high}}$ denote the errors on the low- and high-frequency components, respectively. The detailed hyper-parameter settings are presented in Table 3.

tion $y(x)$ in the frequency domain. The spectral error is defined as:

$$e_{\text{low}}=\sum_{k=-\delta}^{\delta}(\hat{y}_k-\hat{f}_k)^2,\quad e_{\text{high}}=\sum_{|k|>\delta}(\hat{y}_k-\hat{f}_k)^2, \tag{4.2}$$

where $\hat{y}_k$ and $\hat{f}_k$ denote the Fourier coefficients of the predicted and target functions, respectively. For this particular function, the dominant frequencies are located approximately at $k=4$ and $k=25$. Therefore, we choose the spectral cutoff threshold $\delta=15$ to separate low- and high-frequency components.

Fig. 8 illustrates how structured first-layer initialization influences the spectral bias of neural networks. In the baseline training without SFLI, the model exhibits a pronounced

spectral bias: low-frequency components are fitted quickly, while high-frequency components converge slowly and remain underfit for a long duration. Introducing SFLI consistently accelerates convergence across all activation functions, with the most substantial gains observed in the high-frequency region. Remarkably, the Gaussian- and Cosine-based SFLI nearly eliminate the frequency imbalance, achieving comparable accuracy for both low- and high-frequency components. These results indicate that SFLI mitigates spectral bias by encouraging $\varepsilon$-linear independence in neuron activations, allowing the network to capture fine-scale features more efficiently from the outset.

To further evaluate the effectiveness of SFLI in high dimensions, we consider a smooth function defined on the $d$-dimensional hypercube:

**Example 4.4** (High-Dimensional Smooth Function)**.** We consider the high-dimensional function $f(\mathbf{x})=\cos(|\mathbf{x}|^2)$ for $\mathbf{x}\in[-1,1]^d$. The performance of SFLI is tested for different dimensions $d$.

We compare the test relative errors after 20000 training steps obtained with and without SFLI-Gauss method across different input dimensions $d$. For SFLI, we report results using a fixed scaling factor $C=1$ as well as the best performance achieved through a grid search of $C$ over the recommended interval $[0.5,2]$. All experiments use the same network architecture and training configurations to ensure a fair comparison. The results are summarized in Table 1, where each result is averaged over five independent runs. The detailed hyper-parameter settings are presented in Table 3.

As shown in Table 1, the baseline model suffers from a rapid deterioration in accuracy as the input dimension increases. In contrast, SFLI significantly reduces the test error in all cases, even when using a fixed scaling factor $C=1$. Moreover, a simple grid search over $C\in[0.5,2]$ often yields further improvement, indicating that SFLI delivers both robust default performance and enhanced accuracy with minimal tuning.

These results also validate the recommended choice of the shape parameter

$$\gamma=C\frac{n^{\frac{1}{d}}-1}{|\Omega|^{\frac{1}{d}}},$$

which explicitly adapts to the problem dimension $d$ and the layer width $n$, allowing SFLI to maintain stable and effective behavior across varying dimensions. The interval $C\in$

Table 1: Test relative errors in high-dimensional function approximation.

| $d$ | **Baseline** | **SFLI:** $C=1$ | **SFLI:** $C=C^*$ | |
|---|---|---|---|---|
| | Test Rel. Error | Test Rel. Error | $C^*$ | Test Rel. Error |
| 5 | $1.79\times10^{-2}$ | $2.96\times10^{-3}$ | 0.9 | $2.14\times10^{-3}$ |
| 10 | $3.50\times10^{-2}$ | $3.10\times10^{-3}$ | 0.9 | $2.86\times10^{-3}$ |
| 20 | $8.96\times10^{-2}$ | $6.89\times10^{-3}$ | 1.2 | $5.01\times10^{-3}$ |
| 50 | $9.96\times10^{-1}$ | $3.11\times10^{-2}$ | 0.6 | $1.46\times10^{-2}$ |

[0.5,2] provides a practical default range for high-dimensional function approximation tasks.

## 4.2 Solving partial differential equations

In this subsection, we demonstrate the effectiveness of the proposed structured first-layer initialization pre-training method in solving several benchmark partial differential equations (PDEs). Our experiments are based on the Physics-Informed Neural Networks (PINNs) framework [21, 25, 26], which has achieved remarkable success in scientific computing and PDE modeling [2, 3, 5, 9, 13, 15, 38]. Numerical examples follow the training pipeline and hyper-parameter recommendations from [29]. To ensure robustness and competitiveness, we incorporate several state-of-the-art PINN training techniques, including random weight factorization (RWF), periodic boundary condition embedding [4], loss balancing [30, 31], and causal training [28]. We consider both the modified MLP and PirateNet architectures to demonstrate that SFLI is easily integrated with diverse network designs. For simplicity, we present comparisons between baseline models and those using the SFLI-Gaussian variant, which serves as a representative instantiation of our method.

Model training is conducted using mini-batch gradient descent with the Adam optimizer, where collocation points are randomly sampled at each iteration. Exact periodic boundary conditions are enforced in all numerical experiments, whenever applicable, to avoid extra loss constraints. For all examples, we compare the results with and without the application of SFLI-Gaussian, while keeping all other settings—including network architecture and hyper-parameters—identical. Detailed architecture and hyper-parameter settings of the training pipeline are provided in the Appendix A.

**Example 4.5** (Allen–Cahn equation)**.** The Allen–Cahn equation models phase separation in multi-component alloys. We consider the one-dimensional form:

$$\begin{aligned} &u_t - 0.0001u_{xx} + 5u^3 - 5u = 0, \qquad t \in [0,1],\ x \in [-1,1], \\ &u(x,0) = x^2\cos(\pi x), \qquad\qquad\qquad\quad x \in [-1,1]. \end{aligned} \tag{4.3}$$

Fig. 9 shows a comparison between the reference solution and the solution predicted by a trained PirateNet model using SFLI. The agreement between the two solutions demonstrates the model's capability to accurately approximate the dynamics governed by the Allen–Cahn equation.

To systematically evaluate the effectiveness and generality of the proposed SFLI pre-training strategy, we consider four experimental configurations under a unified framework:

- Baseline: the standard PINN model based on the PirateNet architecture, without any advanced techniques.
- SFLI: the Baseline model enhanced solely by the SFLI strategy.

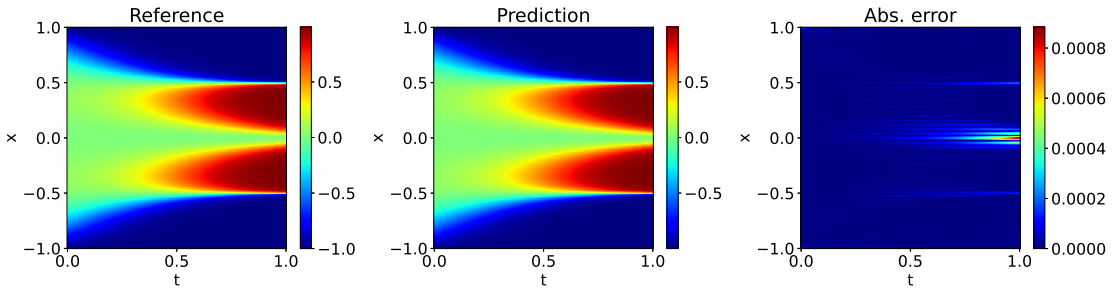


Figure 9: Allen–Cahn equation: Comparison between the solutions predicted by a trained PirateNet with SFLI and the reference solution.

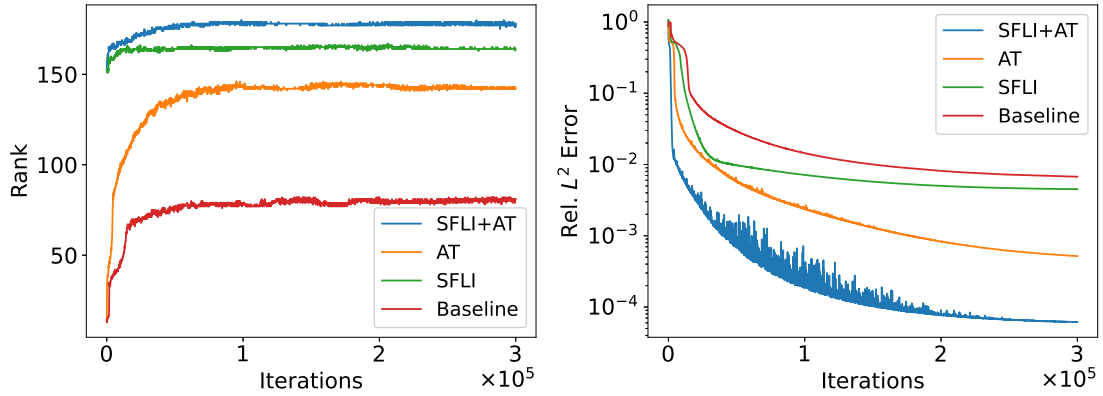


Figure 10: Allen–Cahn equation: Comparison of $\varepsilon$-rank and the relative $L^2$ errors for illustrating the impact of SFLI across different configurations, including Baseline (standard PINN), SFLI, AT (advanced techniques: RWF, loss balancing, causal training), and AT+SFLI. Detailed hyperparameter settings are provided in Table 4.

- AT: the Baseline model enhanced with advanced training techniques (random weight factorization, adaptive loss weighting, and causal training).
- AT+SFLI: the AT model further augmented with SFLI.

All four experiments are conducted using the same PirateNet architecture and identical hyper-parameter settings, ensuring a fair comparison across different configurations. This setup allows us to isolate the effect of SFLI and assess its benefits in two complementary ways: (i) as a standalone enhancement over the standard PINN (Baseline v.s. SFLI), and (ii) as a plug-in improvement to an already advanced setting (AT v.s. AT+SFLI).

In both scenarios, the models with SFLI consistently exhibit higher $\varepsilon$-rank, faster convergence, and lower final errors, demonstrating the effectiveness of SFLI in improving both accuracy and training efficiency.

Overall, SFLI serves as a lightweight yet powerful module that can be seamlessly integrated into various PINN training pipelines to consistently boost performance.

We now consider a classical benchmark problem in computational fluid dynamics: the lid-driven cavity flow. This problem models the steady-state motion of an incompressible viscous fluid confined within a two-dimensional square domain. The governing equations are the incompressible Navier–Stokes equations in non-dimensional form:

**Example 4.6.** (Lid-driven cavity flow)

$$\begin{aligned} \mathbf{u}\cdot\nabla\mathbf{u}+\nabla p-\frac{1}{\mathrm{Re}}\Delta\mathbf{u}=0, \quad (x,y)\in(0,1)^2,\\ \nabla\cdot\mathbf{u}=0, \quad (x,y)\in(0,1)^2. \end{aligned} \tag{4.4}$$

Here, $\mathbf{u}=(u,v)$ denotes the velocity field in $x$ and $y$ directions, and $p$ is the pressure. The top lid moves with a constant horizontal velocity $\mathbf{u}=(1,0)$, while the remaining boundaries are subject to no-slip conditions. We investigate the resulting speed field at Reynolds number $\mathrm{Re}=3200$.

To eliminate corner singularities at the top boundary, we adopt the smoothed auxiliary boundary condition from [27]:

$$\mathbf{u}=\left(1-\frac{\cosh(50(x-0.5))}{\cosh(25)},0\right), \quad \text{for } x\in[0,1],\ y=1. \tag{4.5}$$

Due to the known difficulty of training PINNs at high Reynolds numbers, we employ a curriculum learning strategy to gradually increase the complexity of the problem [16]. Specifically, the model is trained sequentially at increasing Reynolds numbers: $\mathrm{Re}\in\{100, 400, 1000, 1600, 3200\}$, with training iterations allocated as $10^4$, $2\times10^4$, $5\times10^4$, $5\times10^4$, and $5\times10^5$ at each stage, respectively.

Fig. 11 displays the predicted speed field $\sqrt{u^2+v^2}$ for the lid-driven cavity flow at $\mathrm{Re}=3200$ using PirateNet with SFLI. The results show close agreement with the reference solution from [7], with the error primarily localized near the top corners.

To further evaluate the effectiveness of SFLI, Fig. 12 presents a comparison of the $\varepsilon$-rank and relative $L^2$ error between models trained with and without SFLI. The SFLI-enhanced model achieves a substantially higher $\varepsilon$-rank and a much lower relative error (**3.75**% vs. 88.3%), outperforming previous baselines such as JAXPI [29] (15.8%) and the original PirateNet with random Fourier features [27] (4.21%). Notably, the smoothed boundary condition (4.5) introduces an inherent approximation error of 2.59%, suggesting that the observed improvement from 4.21% to 3.75% is a meaningful advance within the resolution limits imposed by the boundary regularization.

In the last example, we aim to demonstrate the effectiveness of our method in simulating incompressible Navier–Stokes flow based on the velocity–vorticity formulation.

**Example 4.7.** (Navier–Stokes flow in a torus)

$$\begin{aligned} w_t+\mathbf{u}\cdot\nabla w=\frac{1}{\mathrm{Re}}\Delta w, \quad \text{in } \Omega\times[0,T],\\ \nabla\cdot\mathbf{u}=0, \quad \text{in } \Omega\times[0,T]. \end{aligned} \tag{4.6}$$

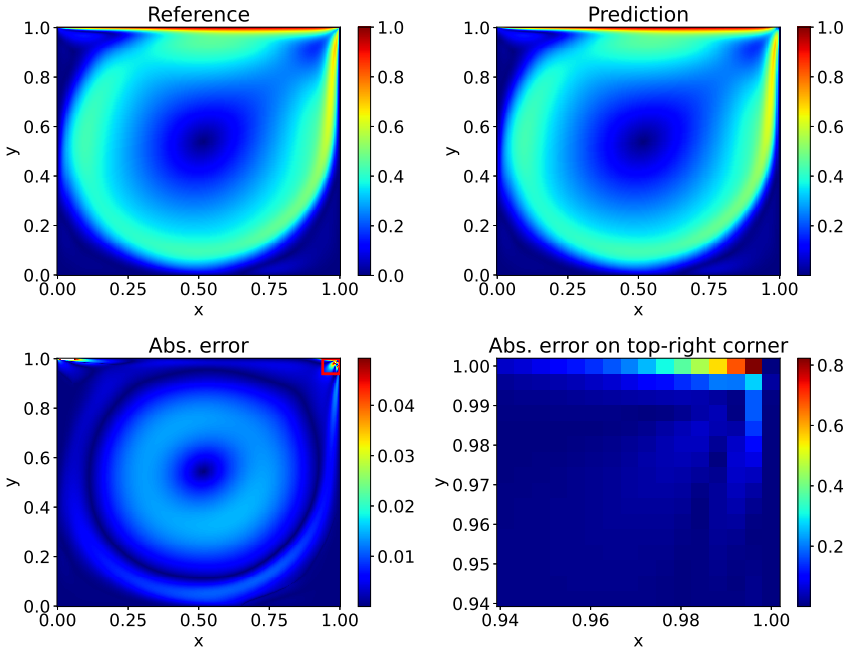


Figure 11: Lid-driven cavity flow at Re=3200: Prediction of speed $\sqrt{u^2+v^2}$ by a trained PirateNet with SFLI. The last graph is the magnified view of the top-right corner of the absolute error, corresponding to the red box in the third plot.

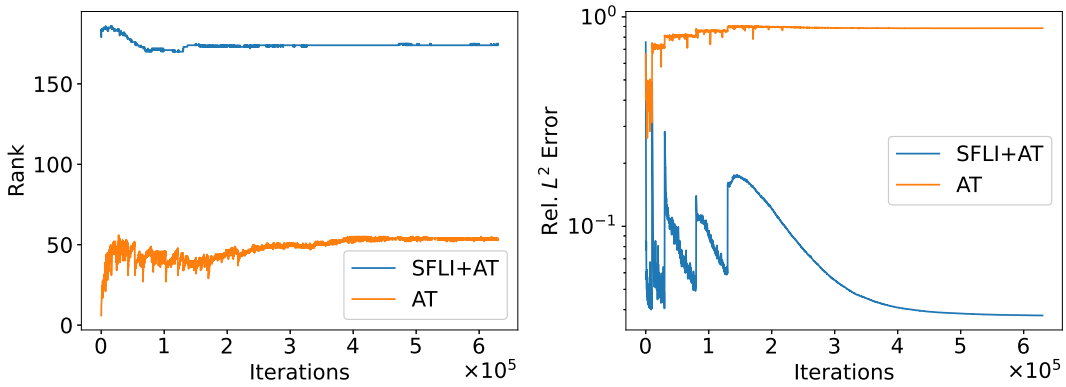


Figure 12: Lid-driven cavity flow at Re=3200: Comparison of $\varepsilon$-rank evolution and the relative $L^2$ errors of speed $\sqrt{u^2+v^2}$ between models with and without SFLI. The final relative $L^2$ errors are **3.75**% and 88.3%. AT: Advanced techniques including RWF, loss balancing and curriculum training. The detailed hyper-parameter settings are presented in Table 5.

Here, $\mathbf{u}=(u,v)$ represents the velocity field, $w=\nabla\times\mathbf{u}$ denotes the vorticity, and Re denotes the Reynolds number. For this example, we set $\Omega=[0,2\pi]^2$, $T=10$, and Re=100. Pe-

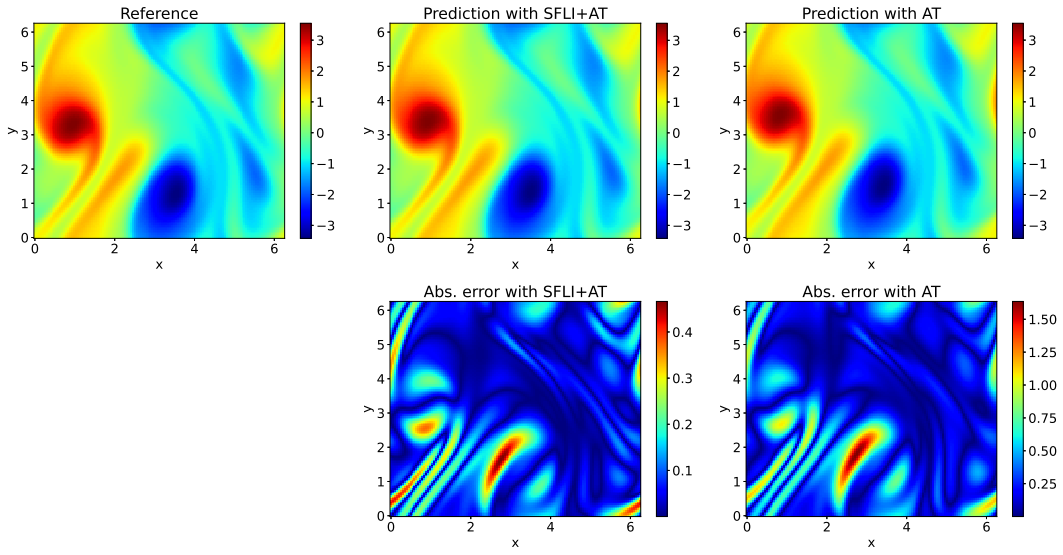


Figure 13: Navier-Stokes flow in a torus: Comparison of the predictions of $w$ at $T=10$ by a modified MLP with and without SFLI. The relative $L^2$ errors are $\mathbf{8.99\times10^{-2}}$ and $2.96\times10^{-1}$. AT: Advanced techniques including RWF, loss balancing, causal training, and time-marching strategy. The detailed hyper-parameter settings are presented in Table 6.

riodic boundary condition is considered and the initial conditions are data from Python package *jaxpi*.

Our objective is to simulate the evolution of the vorticity field up to $T=10$. To this end, the temporal domain is divided into five consecutive intervals, and a time-marching strategy is employed. In each interval, a separate PINN model based on a modified MLP architecture is trained independently. The initial condition for each subsequent interval is provided by the predicted solution at the terminal time of the preceding one.

Fig. 13 compares the predicted solutions of the velocity $w$ at the final time $T=10$ for the Navier–Stokes flow in a torus, with and without the application of SFLI. The inclusion of SFLI leads to a significantly more accurate prediction, as evidenced by the lower relative $L^2$ error of $\mathbf{8.99\times10^{-2}}$, compared to $2.96\times10^{-1}$ in the case without SFLI. This demonstrates that SFLI enhances the neural network's ability to capture complex vortex structures and localized flow features more accurately.

To further assess the effectiveness of SFLI in high-dimensional settings, we consider the following linear reaction-diffusion equation with a known analytical solution.

**Example 4.8.** (High-dimensional parabolic equation)

$$\begin{aligned} u_t-\Delta u+u&=f, \quad \text{in } \Omega\times[0,T],\\ u(\mathbf{x},0)&=u^*(\mathbf{x},0), \quad \text{in } \Omega,\\ u(\mathbf{x},t)&=u^*(\mathbf{x},t), \quad \text{on } \partial\Omega\times[0,T]. \end{aligned} \tag{4.7}$$

Table 2: Test relative errors in solving high-dimensional PDEs.

| Method | $d=5$ | $d=10$ | $d=20$ | $d=50$ |
|---|---|---|---|---|
| **Baseline** | $3.12\times10^{-2}$ | $7.92\times10^{-2}$ | $1.85\times10^{-1}$ | 1.11 |
| **SFLI** | $1.73\times10^{-2}$ | $2.03\times10^{-2}$ | $3.64\times10^{-2}$ | $8.26\times10^{-2}$ |

Here, $\Omega=[-1,1]^d$ is the $d$-dimensional spatial domain and the final time is set to $T=0.2$. The source term is $f(\mathbf{x},t)=2d\mathrm{e}^{-t}\sin(|\mathbf{x}|^2)+4|\mathbf{x}|^2\mathrm{e}^{-t}\cos(|\mathbf{x}|^2)$. The corresponding exact solution is $u^*(\mathbf{x},t)=\mathrm{e}^{-t}\cos(|\mathbf{x}|^2)$.

We compare the relative errors of baseline PINNs and SFLI-enhanced PINNs across increasing spatial dimensions $d=5$, 10, 20, 50. For SFLI, the shape parameter is set according to the recommendation choice (3.3) with a fixed scaling factor $C=1$. The results, averaged over five independent runs, are summarized in Table 2, clearly demonstrating the improved accuracy of SFLI in high-dimensional settings. All experiments use the same network architecture and training configurations to ensure a fair comparison. Each result is averaged over five independent runs. Detailed hyper-parameter settings are provided in Table 7.

As shown in Table 2, the baseline PINNs suffer a severe decline in performance as the spatial dimension increases, eventually failing to converge at $d=50$. In contrast, the SFLI-enhanced model consistently produces stable and accurate results across all tested dimensions. This highlights the robustness of SFLI in enabling effective training even in high-dimensional settings, where standard PINNs may struggle to optimize. Importantly, all results are obtained using a fixed scaling factor $C=1$, without any task-specific tuning.

## 5 Discussion

In this work, we investigated the impact of neural feature diversity on training dynamics through the lens of $\varepsilon$-rank, a quantitative measure of the effective features in neuron functions. Motivated by the insights of the staircase phenomenon, we proposed a structured first-layer initialization pre-training strategy that promotes $\varepsilon$-linear independence among neuron functions in the first hidden layer. The method is activation-function agnostic, architecture compatible, and computationally efficient, making it easy to incorporate into existing network designs. Numerical experiments on high-frequency function approximation and PDE benchmarks, including the Allen–Cahn equation, lid-driven cavity flow, and Navier–Stokes systems, demonstrate that SFLI effectively accelerates convergence and improves predictive accuracy.

For future work, one promising direction is to incorporate $\varepsilon$-rank tracking into the training objective via dynamic regularization. Such a loss design could promote feature diversity throughout the entire training process, rather than only at initialization.

Another consideration is to extend the theoretical analysis of $\varepsilon$-rank to convolutional, attention-based, or graph-based architectures. The current framework focuses primarily on fully connected networks, and its direct extension remains nontrivial.

In summary, this work demonstrates that structural control over the initial representational capacity of neural networks can lead to substantial improvements in training efficiency and accuracy. The proposed SFLI method offers a principled, lightweight, and broadly applicable approach for enhancing neural network performance in scientific computing tasks.

## Acknowledgments

This work is supported by the National Science Foundation of China (No.12271240, 12426312), the fund of the Guangdong Provincial Key Laboratory of Computational Science and Material Design, China (No.2019B030301001), the Shenzhen Natural Science Fund (RCJC20210609103819018), and the Zhuhai Innovation and Entrepreneurship Team Project (2120004000498).

## A Hyper-parameter configurations

This appendix provides the detailed hyper-parameter configurations used in all numerical experiments presented in Section 4. These settings include the network architecture, training strategy, optimization schedules, and shape parameters employed in the SFLI method.

Table 3 summarizes the experimental setups for the function fitting tasks. Tables 4 to 7 detail the complete training configurations for PDE benchmark problems, including the Allen–Cahn equation, lid-driven cavity flow, Navier-Stokes flow in a torus, and the high-dimensional parabolic equation. These configurations are chosen to ensure a fair and consistent comparison between the baseline and SFLI-enhanced models across diverse PDE types and problem scales. For PDE problems, all loss terms are assigned equal weights of 1, or initialized to 1 and adaptively updated by the weighting scheme (Grad Norm or NTK).

Table 3: Numerical experiment settings in function fitting examples.

| **Example** | Layer width | # Layers | Batch size | **Shape parameters** | | | |
|---|---|---|---|---|---|---|---|
| | | | | Gauss | Tanh | Cos | Hat |
| Example 4.1 | 50 | 3 | 201 | - | 15 | - | - |
| Example 4.2 | 100 | 3 | 250 | 4.5 | 4.5 | 4.5 | 4.5 |
| Example 4.3 | 50 | 3 | 201 | 15 | 15 | 15 | 15 |
| Example 4.4 | 128 | 3 | 1000 | Table 1 | | | |

Table 4: (Example 4.5): Allen-Cahn equation.

| Parameter | Value |
|---|---|
| **Architecture Parameters** | |
| Architecture | PirateNet |
| Number of residual blocks | 3 |
| Layer width | 256 |
| Activation | Tanh |
| SFLI shape parameter | $\sqrt{10}$ |
| Random weight factorization | $\mu=1.0$, $\sigma=0.1$ |
| **Learning rate schedule for Adam** | |
| Initial learning rate | $10^{-3}$ |
| Decay rate | 0.9 |
| Decay steps | $5\times10^3$ |
| Warmup steps | $5\times10^3$ |
| **Training** | |
| Training steps | $3\times10^5$ |
| Batch size | 8192 |
| **Weighting** | |
| Weighting scheme | NTK [31] |
| Causal tolerance | 1.0 |
| Number of chunks | 32 |

All function approximation tasks in Table 3 use multilayer perceptrons (MLPs) with `Tanh` activation except the first layer. The shape parameters correspond to the localization scale $\gamma$ in the SFLI scheme (see Equation (3.3)). All models are trained using the Adam optimizer with an initial learning rate of $10^{-3}$, unless otherwise specified.

Table 4 summarizes the detailed hyperparameter configurations for Example 4.5 (Allen–Cahn equation). The reference solution is obtained from the Python package *jaxpi*, which generates the solution by applying a spectral Fourier discretization with 512 modes and a fourth-order exponential time-differencing Runge–Kutta (ETDRK4) scheme with a time step of $10^{-5}$. The validation dataset has a resolution of $201\times512$. The training points for the initial condition correspond to the 512 spatial mesh points, while 8,192 interior points are randomly sampled from the space–time domain at each training iteration. To enforce the periodic boundary condition, the input $(x,t)$ is embedded as $(\cos(\pi x),\sin(\pi x),t)$ after the input layer.

Table 5 summarizes the detailed hyperparameter configurations for Example 4.6 (lid-driven cavity flow). The training points for the boundary condition correspond to 256 mesh points on each side of the cavity. Interior points are uniformly resampled within the domain at every iteration with a batch size of 4,096. The no-slip boundary condition is applied on all walls except for the moving top lid, where the smoothed auxiliary condition (4.5) is adopted to remove corner singularities. All loss terms are weighted equally,

Table 5: (Example 4.6): Lid-driven cavity (Re=3200).

| Parameter | Value |
|---|---|
| **Architecture Parameters** | |
| Architecture | PirateNet |
| Number of residual blocks | 6 |
| Layer width | 256 |
| Activation | Tanh |
| SFLI shape parameter | 10 |
| Random weight factorization | $\mu=1.0$, $\sigma=0.1$ |
| **Learning rate schedule for Adam** | |
| Initial learning rate | $10^{-3}$ |
| Decay rate | 0.9 |
| Decay steps | $10^4$ |
| Warmup steps | $5\times10^3$ |
| **Curriculum Training** | |
| Reynolds Number | [100,400,1000,1600,3200] |
| Training steps | $[10^4,2\times10^4,5\times10^4,5\times10^4,5\times10^5]$ |
| Batch size | 4096 |
| **Weighting** | |
| Weighting scheme | Grad norm [30] |

Table 6: (Example 4.7): Navier-Stokes flow in a torus.

| Parameter | Value |
|---|---|
| **Architecture Parameters** | |
| Architecture | Modified MLP |
| Number of layers | 4 |
| Layer width | 256 |
| Activation | Tanh |
| SFLI shape parameter | $\sqrt{1.5}$ |
| Random weight factorization | $\mu=1.0$, $\sigma=0.1$ |
| **Learning rate schedule for Adam** | |
| Initial learning rate | $10^{-3}$ |
| Decay rate | 0.9 |
| Decay steps | $2\times10^3$ |
| **Time-marching Training** | |
| Number of time windows | 5 |
| Training steps per window | $10^5$ |
| Batch size | 4096 |
| **Weighting** | |
| Weighting scheme | Grad norm |
| Causal tolerance | 1.0 |
| Number of chunks | 32 |

Table 7: (Example 4.8): High-dimensional Parabolic equations.

| Parameter | Value |
|---|---|
| **Architecture Parameters** | |
| Architecture | MLP |
| Number of layers | 4 |
| Layer width | 128 |
| Activation | Tanh |
| SFLI shape parameter | $C=1$ in (3.3) |
| **Learning rate schedule for Adam** | |
| Initial learning rate | $10^{-3}$ |
| Decay rate | 0.9 |
| Decay steps | $2\times10^3$ |
| Training steps | $2\times10^4$ |
| **Batch size** | |
| Interior domain | 512 |
| Initial domain | 256 |
| Boundary domain | $32d$ |
| **Loss weight** | |
| PDE loss | 1.0 |
| Initial loss | 1.0 |
| Boundary loss | 1.0 |

and the gradient-norm-based adaptive weighting scheme is employed during training.

Table 6 summarizes the detailed hyperparameter configurations for Example 4.7 (Navier–Stokes flow in a torus). The training points for the initial condition correspond to the $128\times128$ spatial mesh points, while 8,192 interior points are randomly sampled from the space–time domain at each training iteration. To impose the periodic boundary condition, the input $(x,y,t)$ is embedded as $(\cos(x),\sin(x),\cos(y),\sin(y),t)$ after the input layer. Each time window is trained independently using the time-marching strategy described in Example 4.7, and all loss terms are initialized with equal weights and updated adaptively via the gradient-norm scheme.

Table 7 summarizes the detailed hyperparameter configurations for Example 4.8 (high dimensional parabolic equations). The interior, initial, and boundary sample sizes are listed explicitly in the table, and all corresponding loss weights are set to 1.

## References

[1] Alessandro Achille and Stefano Soatto. Emergence of Invariance and Disentanglement in Deep Representations. *Journal of Machine Learning Research*, 19(50):1–34, 2018.

[2] Santiago Badia, Wei Li, and Alberto F. Martín. Adaptive finite element interpolated neural networks. *Computer Methods in Applied Mechanics and Engineering*, 437:117806, 2025.

[3] Biao Chen, Jifa Zhang, Shuai Zhang, Song Xiaoxiao, and Yao Zheng. Physically guided neural network based on transfer learning (TL-PGNN) for hypersonic heat flux prediction. *Communications in Computational Physics*, 36(3):651–672, 2024.
[4] Suchuan Dong and Naxian Ni. A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks. *Journal of Computational Physics*, 435:110242, 2021.
[5] Vasiliy A. Es'kin, Alexey O. Malkhanov, and Mikhail E. Smorkalov. Are two hidden layers still enough for the physics-informed neural networks? *Journal of Computational Physics*, 537:114085, 2025.
[6] Mario Geiger, Leonardo Petrini, and Matthieu Wyart. Landscape and training regimes in deep learning. *Physics Reports*, 924:1–18, 2021.
[7] U Ghia, K. N Ghia, and C. T Shin. High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *Journal of Computational Physics*, 48(3):387–411, 1982.
[8] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.
[9] Georgios Grekas and Charalambos G. Makridakis. Deep Ritz – Finite element methods: Neural network methods trained with finite elements. *Computer Methods in Applied Mechanics and Engineering*, 437:117798, 2025.
[10] Boris Hanin and David Rolnick. How to start training: The effect of initialization and architecture. In *Advances in Neural Information Processing Systems*, volume 2018-December, pages 571–581. Neural Information Processing Systems Foundation, 2018.
[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.
[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.
[13] Ameya D. Jagtap, and George Em Karniadakis. Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28(5):2002–2041, 2020.
[14] Ameya D. Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020.
[15] Ameya D. Jagtap, Ehsan Kharazmi, and George Em Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, 2020.
[16] Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. In *Advances in Neural Information Processing Systems*, volume 34, pages 26548–26560. Curran Associates, Inc., 2021.
[17] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the Loss Landscape of Neural Nets. In *Advances in Neural Information Processing Systems*, volume 31,

pages 6391–6401. Curran Associates, Inc., 2018.
[18] Jiaqi Li and Xiaodong Yang. A Cyclical Learning Rate Method in Deep Learning Training. In *2020 International Conference on Computer, Information and Telecommunication Systems (CITS)*, pages 1–5, 2020.
[19] Xi'an Li, Jinran Wu, Xin Tai, Jianhua Xu, and You-Gan Wang. Solving a class of multi-scale elliptic PDEs by Fourier-based mixed physics informed neural networks. *Journal of Computational Physics*, 508:113012, 2024.
[20] Ziqi Liu, Wei Cai, and Zhi-Qin John Xu. Multi-Scale Deep Neural Network (MscaleDNN) for Solving Poisson-Boltzmann Equation in Complex Domains. *Communications in Computational Physics*, 28(5):1970–2001, 2020.
[21] Chen Mo, Ding Zhao, Jiao Yuling, Lu Xiliang, Peiying Wu, and Jerry Zhijian, Yang. Convergence analysis of PINNs with over-parameterization. *Communications in Computational Physics*, 37(4):942–974, 2025.
[22] M. S. Nakhodnov, M. S. Kodryan, E. M. Lobacheva, and D. S. Vetrov. Loss Function Dynamics and Landscape for Deep Neural Networks Trained with Quadratic Loss. *Doklady Mathematics*, 106(1):S43–S62, 2022.
[23] Vardan Papyan, X. Y. Han, and David L. Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences*, 117(40):24652–24663, 2020.
[24] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the Spectral Bias of Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019.
[25] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.
[26] Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.
[27] Sifan Wang, Bowen Li, Yuhan Chen, and Paris Perdikaris. PirateNets: Physics-informed deep learning with residual adaptive networks. *Journal of Machine Learning Research*, 25(402):1–51, 2024.
[28] Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality for training physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 421:116813, 2024.
[29] Sifan Wang, Shyam Sankaran, Hanwen Wang, and Paris Perdikaris. An Expert's Guide to Training Physics-informed Neural Networks, 2023.
[30] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.
[31] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why PINNs fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.
[32] Wei Xiong, Xiangyun Long, Stéphane P.A. Bordas, and Chao Jiang. The deep finite element method: A deep learning framework integrating the physics-informed neural networks with the finite element method. *Computer Methods in Applied Mechanics and Engineering*, 436:117681, 2025.
[33] Jinchao Xu. Finite neuron method and convergence analysis. *Communications in Computational Physics*, 28(5):1707–1745, 2020.
[34] Zhi-Qin John Xu. Frequency Principle: Fourier Analysis Sheds Light on Deep Neural Net-

works. *Communications in Computational Physics*, 28(5):1746–1767, 2020.

[35] Zhi-Qin John Xu, Yaoyu Zhang, and Tao Luo. Overview Frequency Principle/Spectral Bias in Deep Learning. *Communications on Applied Mathematics and Computation*, 2024.

[36] Zhi-Qin John Xu, Yaoyu Zhang, and Yanyang Xiao. Training Behavior of Deep Neural Network in Frequency Domain. In Tom Gedeon, Kok Wai Wong, and Minho Lee, editors, *Neural Information Processing*, pages 264–274, Cham, 2019. Springer International Publishing.

[37] Jiang Yang, Yuxiang Zhao, and Quanhui Zhu. Effective Rank and the Staircase Phenomenon: New Insights into Neural Network Training Dynamics. *arXiv preprint arXiv:2412.05144*, 2024.

[38] Jinyong Ying, Xie Yaqi, Jiao Li, and Hongqiao Wang. Accurate adaptive deep learning method for solving elliptic problems. *Communications in Computational Physics*, 37(3):849–876, 2025.

[39] Shijun Zhang, Hongkai Zhao, Yimin Zhong, and Haomin Zhou. Why shallow networks struggle with approximating and learning high frequency: A numerical study. *arXiv preprint arXiv:2306.17301*, 2023.

[40] Zhiwei Zhang, Lulu Zhang, Zhongwang Zhang, and Zhi-Qin John Xu. Loss jump during loss switch in solving PDEs with neural networks. *Communications in Computational Physics*, 36(4):1090–1112, 2024.