SIAM J. SCI. COMPUT.
Vol. 48, No. 1, pp. C77–C102

DEEP COLLOCATION METHOD: A FRAMEWORK FOR SOLVING PDEs USING NEURAL NETWORKS WITH ERROR CONTROL*

MINGXING WENG†, ZHIPING MAO‡, AND JIE SHEN‡

Abstract. Neural networks have shown significant potential in solving PDEs. While deep networks are capable of approximating complex functions, direct one-shot training often faces limitations in both accuracy and computational efficiency. To address these challenges, we propose an adaptive method that uses single-hidden-layer neural networks to construct basis functions guided by the equation residual. The approximate solution is computed within the space spanned by these basis functions, employing a collocation least-squares scheme. As the approximation space gradually expands, the solution is iteratively refined; meanwhile, the progressive improvements serve as reliable a posteriori error indicators that guide the termination of the sequential updates. Additionally, we introduce adaptive strategies for collocation point selection and parameter initialization to enhance robustness and improve the expressiveness of the neural networks. We also derive the approximation error estimate and validate the proposed method with several numerical experiments on various challenging PDEs, demonstrating both high accuracy and robustness of the proposed method.

Key words. neural network, deep collocation method, error estimate, geometric convergence, adaptive method

MSC codes. 68T07, 65N22

DOI. 10.1137/25M1739753

1. Introduction.

PDEs are ubiquitous in modeling various physical, biological, and engineering processes. Traditional numerical methods, such as finite difference methods, finite element methods, and spectral methods, have long been the standard tools for solving PDEs. These techniques have achieved substantial success in scientific computing and have been applied extensively to a wide range of scientific and engineering problems.

However, traditional numerical methods suffer from a significant limitation known as the curse of dimensionality. These methods typically discretize the problem domain into grids or meshes. While effective for low-dimensional problems, the number of grid points required for accurate solutions grows exponentially with dimensions, leading to prohibitive computational storage and cost. Additionally, generating appropriate meshes can be time consuming, particularly for problems with complex geometries. Consequently, mesh-based methods become impractical for high-dimensional problems. Moreover, traditional mesh-based methods lack flexibility, requiring significant structural modifications to accommodate changes in spatial dimensions or PDE types.

To overcome these challenges, machine learning–based methods have emerged as a promising alternative. Neural networks, in particular, provide a mesh-free approach to solving PDEs, effectively bypassing the curse of dimensionality [11] and offering

*Submitted to the journal's Machine Learning Methods for Scientific Computing section March 7, 2025; accepted for publication (in revised form) September 16, 2025; published electronically January 2, 2026.

https://doi.org/10.1137/25M1739753

Funding: This work was supported by the National Natural Science Foundation of China (grants 12171404, 12371409, and W2431008).

†School of Mathematical Sciences, Shanghai Jiao Tong University, Shanghai 200240, People's Republic of China, and School of Mathematical Science, Eastern Institute of Technology, Ningbo, Zhejiang 315200, People's Republic of China (mxweng22@sjtu.edu.cn).

‡School of Mathematical Sciences, Eastern Institute of Technology, Ningbo, Zhejiang 315200, People's Republic of China (zmao@eitech.edu.cn, jshen@eitech.edu.cn).

greater flexibility in handling complex geometries. Techniques like physics-informed neural networks (PINNs) [21, 16], the deep Ritz method [27], weak adversarial networks [32], and the deep Galerkin method [22] have gained popularity for their ability to approximate PDE solutions without the need for predefined meshes. *However, despite their potential applications in engineering, machine learning methods still face limitations in terms of accuracy and computational efficiency.* Training neural networks involves solving a highly nonlinear and nonconvex optimization problem, which can be particularly challenging when high accuracy is needed. In addition, the convergence is usually not guaranteed.

In recent years, researchers have introduced various strategies to enhance the accuracy and efficiency of neural network–based solvers for PDEs. A significant focus has been on improving PINNs with a wide range of advancements, including adaptive sampling techniques for handling sharp solutions [17, 10, 18] or high-dimensional problems [23, 34], homogenization methods for multiscale PDEs [15], and second-order optimization methods to support multitask learning [24], among others. Beyond the PINN framework, two particularly noteworthy high-accuracy methods include the randomized neural network–based approaches [13, 8, 6, 33, 7], which aim to reduce optimization errors and simplify training, and the residual-driven iterative methods [1, 26, 4, 28], which progressively refine solutions through successive approximations.

Randomized neural networks (RNNs) [19, 14] offer a promising approach to reduce the optimization error introduced by training algorithms, albeit at the cost of some approximation capability. Specifically, RNNs simplify the training process by fixing certain parameters and focusing optimization on the linear coefficients before the output layer, reducing the problem to a least-squares solution of a linear system. A prominent example is the extreme learning machine (ELM) [13], a single-hidden-layer RNN that has been successfully applied to solve PDEs in strong form [31, 9, 25]. Building on ELM, Dong and Li introduced the local extreme learning machine (locELM) [8], which integrates ELM with domain decomposition, using a local feedforward network for each subdomain. This method strikes an effective balance between computational efficiency and high-accuracy solutions. The random feature method (RFM), introduced by Chen et al. [6], extends locELM by replacing domain decomposition with the partition of unity technique. RFM incorporates multiscale basis functions into the network and includes adaptive weight scaling in the loss function, achieving near-spectral convergence in experiments, including a challenging problem in porous geometry.

The aforementioned RNN-based methods deliver high accuracy for smooth solutions, but they struggle with problems having singularities or sharp interfaces. In addition, both locELM and RFM *heavily rely on the initialization of hidden-layer parameters*. To mitigate the need for manual tuning, Zhang et al. [33] proposed a transferable hidden-layer parameter pretraining method that enhances network expressiveness by fitting randomly generated Gaussian random fields. Based on a similar initialization strategy, Dang and Wang [7] developed the adaptive growing randomized neural network (AG-RNN), which dynamically adjusts the network size throughout the iterative solution process. By constructively adding neurons based on the residuals from previous steps, AG-RNN enhances its capacity to represent complex solutions without introducing excessive additional optimization steps. AG-RNN has demonstrated high accuracy in solving PDEs with sharp or discontinuous solutions, offering an efficient and adaptive framework for tackling challenging problems.

These RNN-based methods offer a sophisticated blend of the representational power of neural networks and therobustness of traditional numerical techniques. By

constructing approximation spaces with neural networks and applying least-squares methods during training, they strike a balance between computational efficiency and solution accuracy. Nevertheless, all the aforementioned RNN-based methods have the fundamental issue of *ill-conditioning*.

Beyond RNN-based approaches, residual-based methods have also garnered considerable interest. By minimizing the residuals at each step, these methods achieve progressive improvement in the solution, often leading to highly accurate results. Galerkin neural networks (GNNs), developed by Ainsworth and Dong [1, 2], proposed a residual-driven iterative scheme in the weak form of PDEs. This method provides an error estimator as a stopping criterion, ensuring high accuracy. Extended GNNs [3] further generalize this approach to handle broader boundary value problems, particularly improving accuracy in the presence of singularities. However, a key limitation of GNN is its reliance on the mesh-based Gaussian quadrature for numerical integration, which significantly increases computational cost and leads to the curse of dimensionality in high-dimensional problems. In addition, GNN faces challenges in handling nonlinear problems and problems in complex domains. In contrast to GNN, which works with the weak form of PDEs, multistage neural networks (MSNNs) by Wang and Li [26] and multilevel neural networks (MLNNs) by Aldirany et al. [4] apply residual corrections in the strong form of PDEs. Both methods introduce new networks at each iteration, specifically tailored to reduce the residual of the previous stage. With appropriate initialization, these approaches have achieved substantial error reduction, reaching machine precision in some cases. Another development is the multigrade neural network (MGNN) proposed by Xu [28] and Xu and Zeng [29]. MGNN deepens the network structure with each iteration while preserving fixed parameters from earlier stages, simplifying optimization and enhancing accuracy. Nevertheless, its training process exhibits inconsistent error reduction, making convergence rates less predictable. Furthermore, MSNN, MLNN, and MGNN all suffer from prolonged training times due to their reliance on the Adam optimizer, which requires a large number of optimization steps.

The aim of this work is to propose a collocation framework for solving a wide range of PDEs using neural networks with error control. In particular, we follow the idea of GNN by greedily constructing a sequence of finite-dimensional subspaces where the basis functions are generated through a series of neural networks. Our approach incorporates adaptive strategies for network initialization and operates within a collocation framework, leveraging a least-squares formulation for network training. This design significantly enhances *computational efficiency*, *accuracy*, and *flexibility*.

The proposed method enjoys several key advantages:

- *Robustness, efficiency, and high accuracy.* We integrate several adaptive strategies to reduce the difficulty in training neural networks and alleviate the challenges of hyperparameter tuning. Moreover, the method maintains stable error decay when hyperparameters are chosen within a suitable range, ensuring *robustness*, *efficiency*, and *accuracy*.
- *Guaranteed convergence.* We establish a convergence property demonstrating that the error remains nonincreasing throughout iterations. Under appropriate network optimization assumptions, we derive a reliable a posteriori error estimator and provide an error estimate, offering theoretical insights into achieving *geometric convergence*. These findings are further validated by numerical experiments.
- *Broader applicability.* Compared with GNN, the proposed collocation-based method exhibits broader applicability across various problems, such as *nonlinear problems* and *problems in complex domains*.

The remainder of this paper is organized as follows. In section 2, we introduce the proposed methodology, outlining the collocation framework, the construction of neural network basis functions, and the adaptive strategies used for the selection of collocation points and network initialization. Section 3 is dedicated to proving the convergence of the method as well as providing an a posteriori indicator to guide the termination of the iterative process. In section 4, we present numerical experiments that assess the performance of the proposed method across a range of PDEs. Finally, in section 5, we summarize our findings and suggest possible avenues for future research.

2. Methodology. In this section, we present the fundamental framework of our algorithm. The core idea is to construct a set of adaptive basis functions using single-hidden-layer feedforward neural networks (SLFNs) and solve PDEs through a collocation scheme. Subsection 2.1 introduces the linear PDE setting and the corresponding collocation framework, followed by an outline of the overall algorithmic procedure. Subsequently, subsections 2.2 and 2.3 describe, respectively, the training process for constructing neural network basis functions and the adaptive strategies designed to enhance the solver performance.

2.1. Problem statement and collocation scheme. Consider the following linear PDE defined on a bounded domain $\Omega \subset \mathbb{R}^d$ with boundary $\partial\Omega$:

$$\mathcal{L}u = f \quad \text{in } \Omega \quad \text{and } \mathcal{B}u = g \quad \text{on } \partial\Omega, \tag{2.1}$$

where $\mathcal{L}$ is a linear differential operator, $\mathcal{B}$ is a linear boundary operator, and f and g are the source term and prescribed boundary data, respectively.

For the collocation method, the PDE and boundary conditions are enforced at discrete points within the domain and on the boundary. In particular, let the collocation points for the domain and boundary be $X_\Omega = \{\boldsymbol{x}_m\}_{m=1}^{M_\Omega}$ and $X_{\partial\Omega} = \{\boldsymbol{x}'_m\}_{m=1}^{M_{\partial\Omega}}$, respectively. Given a set of basis functions $\Phi = \{\phi_n\}_{n=1}^{N}$, let the approximated solution be given by $u_N(\boldsymbol{x}) = \sum_{n=1}^{N} \beta_n \phi_n(\boldsymbol{x})$. Then the resulting least-squares system is given by $\boldsymbol{A}\boldsymbol{\beta} = \boldsymbol{b}$, where $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_N]^T$ and

$$\boldsymbol{A} = \begin{pmatrix} \mathcal{L}\phi_n\left(\boldsymbol{x}_m\right)_{m,n=1}^{M_\Omega,N} \\ \lambda\phi_n\left(\boldsymbol{x}'_m\right)_{m,n=1}^{M_{\partial\Omega},N} \end{pmatrix} \in \mathbb{R}^{M\times N}, \quad \boldsymbol{b} = \begin{pmatrix} f\left(\boldsymbol{x}_m\right)_{m=1}^{M_\Omega} \\ \lambda g\left(\boldsymbol{x}'_m\right)_{m=1}^{M_{\partial\Omega}} \end{pmatrix} \in \mathbb{R}^{M}.$$

The upper block of $\boldsymbol{A}$ corresponds to enforcing the PDE in the interior domain, while the lower block enforces the boundary conditions with penalty parameter $\lambda \in \mathbb{R}^+$. We define this process as a function

$$\boldsymbol{\beta} = \textsc{CollocationLSQ}(\mathcal{L}, f, \mathcal{B}, g, \Phi, X_\Omega, X_{\partial\Omega}, \lambda).$$

For traditional numerical approaches, e.g., Fourier series or polynomial expansions, the choice of basis functions is typically prescribed. While effective in many scenarios, these methods may become inadequate when trying to resolve localized features or steep gradients. To overcome these limitations, we propose to construct adaptive basis functions using neural networks by employing a greedy procedure which allows more flexibility in capturing complex solution behaviors directly from the residual of the PDEs.

To facilitate a global understanding of the proposed methodology, Algorithm 2.1 presents the overall PDE-solving framework, referencing key subalgorithms detailed

Algorithm 2.1. Main algorithm.

Data: Equation information $(\mathcal{L}, f, \mathcal{B}, g, \Omega)$ from (2.1), parameter of scaling factor (k, b), initial guess u_0 (typically zero), initial network width N_0, number of interior and boundary collocations $(M_\Omega, M_{\partial\Omega})$, regularization parameter λ, maximum optimization steps N_{opt}, learning rate α, tolerance ϵ

Result: The numerical solution u_s

Generate $2M_\Omega/3$ uniform points in the domain X_Ω^{I}, and $M_{\partial\Omega}$ points on the boundary $X_{\partial\Omega}$. Set the initial basis $\psi_0 = u_0$, stage number $s = 0$ **while** $s = 0$ *or* $|||u_s - u_{s-1}||| > \epsilon$ **do**

  Set $s \leftarrow s + 1$ Sampling new residual-based points $X_\Omega^{\text{II},s}$ by *Rejection Sampling* (see Algorithm 2.3) with inputs $(D(x), N) = (|f_{s-1}^{\text{res}}(\boldsymbol{x})|, M_\Omega/3)$. Set $X_\Omega^s = X_\Omega^{\text{I}} \cup X_\Omega^{\text{II},s}, X_{\partial\Omega}^s = X_{\partial\Omega}$ Constructing a new neural network basis ψ_s by *Adaptive Initialization* (see Algorithm 2.4) with inputs $(N_s, R_s) = (2^s N_0, ks+b)$. Let Φ_s denote the set of the hidden layer functions of ψ_s Training the basis ψ_s to approximate the residual equation (2.2) using *Training of Adaptive Basis* (see Algorithm 2.2) with inputs $(\Phi_s, \psi_s, X_\Omega^s, X_{\partial\Omega}^s, \lambda, N_{\text{opt}}, \alpha)$ Updating the approximation u_s by CollocLSQ in the augmented space $\text{span}\{\psi_i\}_{i=0}^s$ (omit ψ_0 if it equals zero)

return u_s

in subsequent subsections. This structure not only clarifies how each component integrates into the full solution process but also highlights several adaptive mechanisms designed to alleviate the challenges of hyperparameter tuning, which is often experience-driven and sensitive in PINN-based solvers. Specifically, (1) network width is progressively increased and automatically terminated via the proposed a posteriori error estimator (see Proposition 3.3), eliminating manual selection; (2) optimizer sensitivity is reduced, as Adam is used only for refinement after the collocation least-squares stage (Algorithm 2.2), with a coarse learning rate typically sufficient; and (3) collocation points are adaptively sampled based on the residual (Algorithm 2.3), allowing accurate approximation with a moderate number of points.

2.2. Construction of neural network basis. We construct each basis function using an SLFN, which allows for efficient training using a hybrid strategy detailed in this section.

As demonstrated in the ELM framework [13], SLFNs can be trained efficiently by fixing the weights and biases of the hidden layer and solving a linear least-squares problem to determine the output layer coefficients. While this approach reduces the optimization complexity, it may limit the expressiveness of the network. To enhance the representational capacity of the network, we use an *adaptive initialization* strategy based on the residual of the PDE and apply a small number of Adam optimization steps after solving the least-squares problem to refine the hidden-layer parameters.

An SLFN with N hidden neurons is mathematically expressed as

$$f_{NN}(\boldsymbol{x}) = \sum_{i=1}^{N} c_i \, \sigma\big(\boldsymbol{w}_i \cdot \boldsymbol{x} + b_i\big), \quad \boldsymbol{x} \in \Omega \subset \mathbb{R}^d,$$

where $c_i \in \mathbb{R}$ are the output layer coefficients, σ is the activation function, and $\boldsymbol{w}_i \in \mathbb{R}^d$, $b_i \in \mathbb{R}$ represent the weights and biases of the hidden neurons, respectively. We define

Algorithm 2.2. Training of adaptive basis.

Data: Residual equation information $(\mathcal{L}, f_{s-1}^{\text{res}}, \mathcal{B}, g_{s-1}^{\text{res}})$ from (2.2), new hidden layer functions Φ_s and new basis ψ_s with parameters $\theta_s = (\boldsymbol{W}_s, \boldsymbol{b}_s, \boldsymbol{c}_s)$, collocation points $X_\Omega^s, X_{\partial\Omega}^s$, regularization parameter λ, maximum optimization steps N_{opt}, learning rate α

Result: Updated basis function $\psi_s(\boldsymbol{x})$

Compute the output layer coefficients $\boldsymbol{c}_s = \text{CollоLSQ}(\mathcal{L}, f_{s-1}^{\text{res}}, \mathcal{B}, g_{s-1}^{\text{res}}, \Phi_s, X_\Omega^s, X_{\partial\Omega}^s, \lambda)$

for $n = 1$ **to** N_{opt} **do**

  Calculate the loss function $L(\boldsymbol{W}_s, \boldsymbol{b}_s)$ as in (2.3) Update the θ_s by Adam with loss $L(\boldsymbol{W}_s, \boldsymbol{b}_s)$ and learning rate α

Update the output layer coefficients $\boldsymbol{c}_s = \text{CollоLSQ}(\mathcal{L}, f_{s-1}^{\text{res}}, \mathcal{B}, g_{s-1}^{\text{res}}, \Phi_s, X_\Omega^s, X_{\partial\Omega}^s, \lambda)$

return ψ_s

$\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_N]$, $\boldsymbol{b} = (b_1, b_2, \ldots, b_N)$, and $\boldsymbol{c} = (c_1, c_2, \cdots, c_N)$, encapsulating the network parameters as $\theta = \{\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{c}\}$.

Starting with an initial approximation u_0 (typically zero) and the initial basis function $\psi_0 := u_0$, we iteratively construct adaptive basis functions using SLFNs. Following the multistage training strategy outlined in [26], we refer to the construction of each basis function as a stage. At stage $s > 0$, given the current approximation u_{s-1} and basis functions $\Psi_{s-1} = \{\psi_i\}_{i=0}^{s-1}$ (omit ψ_0 if it equals zero), we initialize a new SLFN basis ψ_s with width N_s and parameters $\theta_s = \{\boldsymbol{W}_s, \boldsymbol{b}_s, \boldsymbol{c}_s\}$. The hidden-layer parameters $(\boldsymbol{W}_s, \boldsymbol{b}_s)$ are adaptively initialized (detailed in subsection 2.3) to ensure flexibility in capturing the key features of the residual, which is defined as

$$\begin{aligned} \mathcal{L}e_{s-1}(\boldsymbol{x}) &= f_{s-1}^{\text{res}}(\boldsymbol{x}) := f(\boldsymbol{x}) - \mathcal{L}u_{s-1}(\boldsymbol{x}) \quad \text{in } \Omega, \\ \mathcal{B}e_{s-1}(\boldsymbol{x}) &= g_{s-1}^{\text{res}}(\boldsymbol{x}) := g(\boldsymbol{x}) - \mathcal{B}u_{s-1}(\boldsymbol{x}) \quad \text{on } \partial\Omega, \end{aligned} \tag{2.2}$$

where $e_s(\boldsymbol{x}) = u(\boldsymbol{x}) - u_s(\boldsymbol{x})$ represents the error after stage s.

To train $\psi_s(\boldsymbol{x})$, we begin by fixing the hidden-layer weights and biases, treating the SLFN as a linear combination of hidden-layer functions $\Phi_s = \{\phi_{s,i}(\boldsymbol{x}) = \sigma(\boldsymbol{w}_{s,i} \cdot \boldsymbol{x} + b_{s,i})\}_{i=1}^{N_s}$. Next, we select the collocation points X_Ω^s and $X_{\partial\Omega}^s$ (as detailed in subsection 2.3) and apply the CollоLSQ function to compute the output layer coefficients $\boldsymbol{c}_s$. The hidden-layer parameters are then refined using the Adam optimizer to enhance the network performance, with the loss function defined as

$$L(\boldsymbol{W}_s, \boldsymbol{b}_s) = \sum_{\boldsymbol{x} \in X} \left| \mathcal{L}\psi_s(\boldsymbol{x}) - f_{s-1}^{\text{res}}(\boldsymbol{x}) \right|^2 + \lambda^2 \sum_{\boldsymbol{x} \in X_{bc}} \left| \mathcal{B}\psi_s(\boldsymbol{x}) - g_{s-1}^{\text{res}}(\boldsymbol{x}) \right|^2. \tag{2.3}$$

Once the optimization process is completed, the output layer coefficients $\boldsymbol{c}_s$ are updated once again using CollоLSQ. The complete training procedure is provided in Algorithm 2.2.

2.3. Adaptive strategies. As discussed in the previous section, SLFNs can achieve efficient training with low optimization errors. However, this efficiency often comes at the cost of reduced network expressiveness. Therefore, it becomes essential to adaptively initialize the hidden-layer parameters to effectively capture the critical features of the target function. Moreover, the selection of training points plays an important role in maintaining both the efficiency and the robustness of the training process. In this section, we present adaptive strategies for both collocation point selection and parameter initialization to enhance the network performance.

Algorithm 2.3. Rejection sampling.

Data: Sampling distribution function $D(\boldsymbol{x})$, number of points N
Result: Adaptive collocation points X
Set $M = \max_{\boldsymbol{x}\in\Omega} |D(\boldsymbol{x})|$ $X \leftarrow \emptyset$ **while** $|X| < N$ **do**
  Sample a candidate point $\boldsymbol{x} \sim \mathrm{U}(\Omega)$ Sample a random number $r \sim \mathrm{U}(0, M)$ **if** $|D(\boldsymbol{x})| \geq r$ **then**
    $X \leftarrow X \cup \{\boldsymbol{x}\}$
return X

2.3.1. Adaptive collocations. For smooth target functions with minimal variation, equidistant or uniform collocation is generally sufficient to achieve reliable approximations. However, in cases where the solution exhibits sharp gradients, singularities, or other localized features, adaptive sampling techniques based on the equation residual or gradient typically provide improved accuracy and stability.

At each stage $s \geq 1$, we divide the interior collocation points into two subsets. The first subset, X_Ω^{I}, contains equidistantly sampled points, accounting for two-thirds of the total, to ensure uniform coverage of the domain. The remaining one-third, denoted by $X_\Omega^{\mathrm{II},s}$, is selected adaptively to improve accuracy in regions with larger residuals.

To this end, we employ *rejection sampling* [5], a Monte Carlo technique for generating samples from a complex target distribution. It operates by randomly selecting points from a region that encloses the target density and accepting only those that fall beneath its graph. The detailed sampling procedure is summarized in Algorithm 2.3. In our case, samples in $X_\Omega^{\mathrm{II},s}$ are drawn according to the residual magnitude, i.e., $D(\boldsymbol{x}) = |f_{s-1}^{\mathrm{res}}(\boldsymbol{x})|$. By assigning a higher likelihood of selection to points with larger residuals, we focus on refining the areas most in need. Boundary collocation points are sampled equidistantly throughout to ensure consistent enforcement of boundary conditions.

To handle sampling in a complex geometrical domain $\widetilde{\Omega}$, we employ an indicator function to determine whether a candidate point lies within the domain:

$$\chi_{\widetilde{\Omega}} = \begin{cases} 1, & \boldsymbol{x} \in \widetilde{\Omega}, \\ 0, & \boldsymbol{x} \notin \widetilde{\Omega}. \end{cases}$$

As in the regular domain case, we perform *rejection sampling* (see Algorithm 2.3) over a bounding box Ω that encloses $\widetilde{\Omega}$. The only difference is that we define the sampling distribution as $D(\boldsymbol{x}) = \chi_{\widetilde{\Omega}}(\boldsymbol{x})\,|f_{s-1}^{\mathrm{res}}(\boldsymbol{x})|$, ensuring that only points within $\widetilde{\Omega}$ are retained. This approach allows for efficient handling of irregularly shaped or disconnected domains without requiring explicit parameterization of the boundaries.

2.3.2. Adaptive initialization. The initialization of network parameters is crucial for the convergence and accuracy of the solution. Both the weights and the biases determine the underlying approximating subspace of the SLFN. To enhance the network capacity to capture complex features, we propose an adaptive initialization strategy that leverages the equation residual to guide the selection of hidden-layer parameters.

To achieve effective initialization, it is essential to understand how the parameters influence the network capacity. For an SLFN with fixed hidden-layer parameters $(\boldsymbol{W}, \boldsymbol{b})$, its approximation subspace consists of functions of the form

$$\phi_i(\boldsymbol{x}) = \sigma(\boldsymbol{w}_i \cdot \boldsymbol{x} + b_i), \quad i = 1, 2, \ldots, N.$$

For the most commonly used activation functions, such as sigmoid, tanh, Gaussian, ReLU, and so on, the weight $\boldsymbol{w}_i$ controls the slope or frequency of ϕ_i, and ϕ_i exhibits the most significant feature around the hyperplane:

$$\{\boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{w}_i \cdot \boldsymbol{x} + b_i = 0\}.$$

We refer to this hyperplane as the partition hyperplane of ϕ_i following [33]. Both the frequency and the location of the hyperplane are crucial for capturing the features of the target function.

Since weights are the only parameters that control the frequency of basis functions, we first consider their initialization. Leveraging the iterative approximation process, we initialize the weights at each stage s using a scaling factor R_s to emphasize different frequency components of the target function. Specifically, the weights are drawn from a uniform distribution $\boldsymbol{W}_s \sim \mathbb{U}(-R_s, R_s)$.

Neural networks are known to exhibit spectral bias [20] or the frequency principle [30], which means that they tend to learn low-frequency features first. From this perspective, the scaling factors $\{R_s\}$ are designed to be nondecreasing, allowing the network to progressively capture higher-frequency components during training. This strategy enhances the ability of networks to represent solutions with varying spectral characteristics, effectively addressing challenges in multiscale phenomena.

With the weights fixed, we determine the biases $b_{s,i}$, which influence the partition hyperplane of $\phi_{s,i}(\boldsymbol{x})$. As in [33], we define the hyperplane density at $\boldsymbol{x}$ as the percentage of neurons whose partition hyperplane intersects the ball with center $\boldsymbol{x}$ and radius τ:

$$D_\tau(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\left(\frac{|\boldsymbol{w}_i \cdot \boldsymbol{x} + b_i|}{\|\boldsymbol{w}_i\|_2} \le \tau \right).$$

It is reasonable to consider that if the hyperplanes pass through the points where the target function is largest in magnitude, then the network will have good approximation capability. Thus, we employ the rejection sampling algorithm to select a set of base points $X^s_{base} = \{\boldsymbol{x}_{s,i}\}_{i=1}^{N_s}$ based on the equation residual f^{res}_{s-1}. We then initialize the biases $\boldsymbol{b}_s = (b_{s,1}, b_{s,2}, \cdots, b_{s,N_s})$ such that the partition hyperplane of $\phi_{s,i}$ passes through the base points:

$$b_{s,i} = -\boldsymbol{w}_{s,i} \cdot \boldsymbol{x}_{s,i}, \quad i = 1, 2, \ldots, N_s. \tag{2.4}$$

This initialization enhances the expressiveness of the neural network at key base points, thereby improving its ability to capture local features and increasing approximation accuracy in regions with large residuals. To illustrate the effectiveness of this approach, we use $|\sin(\pi x)\cos(\pi y)|$ as the target distribution. Figure 1 presents the resulting density distributions of the hyperplanes initialized using (2.4). The results demonstrate that the proposed method achieves an appropriately concentrated distribution.

In addition to parameter initialization, the selection of an appropriate network width N_s also plays a key role in ensuring algorithm convergence. Rather than discussing it here, we defer this topic to the next section, where the analysis provides essential guidance. A detailed strategy for choosing the network width can be found in Remark 3.5. Finally, the complete adaptive initialization strategy is summarized in Algorithm 2.4.

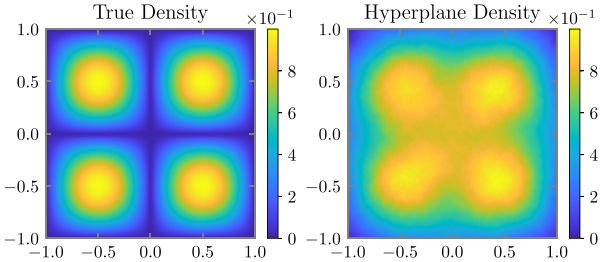

FIG. 1. *An example of biases adaptive initialization. Left: The target density* $|\sin(\pi x)\cos(\pi y)|$. *Right: The density distribution of* 20,000 *initialized hyperplanes computed with* $\tau = 0.1$.

Algorithm 2.4. Adaptive initialization.

Data: Equation residual f_{s-1}^{res}, number of hidden neurons N_s, scaling factor R_s
Result: Hidden layer parameters $(\boldsymbol{W}_s, \boldsymbol{b}_s)$
Function AdapInit(f_{s-1}^{res}, N_s, R_s)**:**
  Initialize the weights $\boldsymbol{W}_s \sim \mathbb{U}(-R_s, R_s)$ Select base points $X_{base}^s = \{\boldsymbol{x}_{s,i}\}_{i=1}^{N_s}$ by *Rejection Sampling* (see Algorithm 2.3) with sampling distribution function $D(\boldsymbol{x}) = |f_{s-1}^{\text{res}}(\boldsymbol{x})|$ Initialize the biases $b_{s,i} = -\boldsymbol{w}_{s,i} \cdot \boldsymbol{x}_{s,i}$ **return** $(\boldsymbol{W}_s, \boldsymbol{b}_s)$

Remark 2.1. Regarding the selection strategy for the scaling factors, Chen et al. [6] explored an approach based on spectral analysis to determine R_s, which works well when using sine or cosine as the activation function. However, for most activation functions, we have not yet found a deterministic rule to select R_s. Nevertheless, based on our empirical observations, a linear scheme of the form $R_s = ks + b$, where $k \in \mathbb{N}$ and $b \in \mathbb{Z}$ are constants, generally leads to good performance.

3. Numerical analysis. In this section, we present a convergence analysis for the proposed method. Specifically, we give the approximation error estimate. Furthermore, under appropriate optimization assumptions, we derive an a posteriori error indicator to guide the termination of the iterative process.

To this end, we consider the linear boundary value problem (2.1) and introduce the associated norm defined by

$$|||u||| = \left(\|\mathcal{L}u\|^2_{L^2(\Omega)} + \|\lambda \mathcal{B}u\|^2_{L^2(\partial\Omega)} \right)^{\frac{1}{2}},$$

where $\lambda \in \mathbb{R}^+$. Obviously, $|||\cdot|||$ is indeed a norm. The linearity property follows immediately. The positive definiteness is guaranteed by the existence and uniqueness of the PDE solution, while the triangle inequality is derived using the Cauchy–Schwarz inequality. Note that in the special case of fitting problems, which can be interpreted as PDEs with the identity operator $\mathcal{L}$ and no boundary operator, the norm $|||\cdot|||$ is actually the L^2 norm.

Next, we prove that the error at each stage decreases with respect to the norm $|||\cdot|||$. At stage s, we project the exact solution u onto the subspace spanned by $\Psi_s = \{\psi_i\}_{i=0}^s$ in the sense of the norm $|||\cdot|||$. Although our algorithm employs a discrete projection in the collocation sense, the analysis considers a continuous projection given by

$$u_s^* = \underset{v \in \operatorname{span}\Psi_s}{\arg\min} |||u - v||| := P_{\Psi_s} u.$$

The errors and the improvements between consecutive stages are defined as

$$e_s^* = u - u_s^*, \quad v_s^* = u_s^* - u_{s-1}^*. \tag{3.1}$$

By the properties of the projection, it is clear that $|||e_{s+1}^*||| \le |||e_s^*|||$. This suggests that the error cannot increase at each stage. A more detailed analysis shows that, except when the solutions at consecutive stages are identical, the error strictly decreases. The following proposition formally establishes this behavior.

PROPOSITION 3.1. *Consider the linear boundary value problem* (2.1) *with a source term* $f \in L^2(\Omega)$ *and boundary data* $g \in L^2(\partial\Omega)$. *The error defined in* (3.1) *either strictly decreases at each stage* $s \in \mathbb{N}^+$, *i.e.,* $|||e_{s+1}^*||| < |||e_s^*|||$, *or remains unchanged if and only if* $u_{s+1}^* = u_s^*$.

Proof. Direct calculation gives

$$|||e_s^*|||^2 = \left|||e_{s+1}^*\right|||^2 + \left|||v_{s+1}^*\right|||^2 + 2\left(\langle \mathcal{L}e_{s+1}^*, \mathcal{L}v_{s+1}^*\rangle_\Omega + \lambda^2 \langle \mathcal{B}e_{s+1}^*, \mathcal{B}v_{s+1}^*\rangle_{\partial\Omega}\right).$$

We claim that

$$\langle \mathcal{L}e_{s+1}^*, \mathcal{L}v_{s+1}^*\rangle_\Omega + \lambda^2 \langle \mathcal{B}e_{s+1}^*, \mathcal{B}v_{s+1}^*\rangle_{\partial\Omega} \ge 0. \tag{3.2}$$

Assuming that (3.2) holds, it follows that

$$|||e_s^*|||^2 \ge \left|||e_{s+1}^*\right|||^2 + \left|||v_{s+1}^*\right|||^2 \ge \left|||e_{s+1}^*\right|||^2.$$

The equality holds if and only if $|||v_{s+1}^*||| = 0$, which implies that $v_{s+1}^* = u_{s+1}^* - u_s^* = 0$.

It remains to prove (3.2). Suppose, for the sake of contradiction, that (3.2) does not hold. Define $v_{s+1}^\eta = (1-\eta)v_{s+1}^* \in \operatorname{span}\Psi_{s+1}$. Then we have

$$\begin{aligned}\left|||e_s^* - v_{s+1}^\eta|||\right|^2 =& \left|||e_s^* - v_{s+1}^*|||\right|^2 + \eta\big(\eta\left|||v_{s+1}^*\right|||^2 \\ &+ 2\langle \mathcal{L}v_{s+1}^*, \mathcal{L}e_{s+1}^*\rangle_\Omega + 2\lambda^2\langle \mathcal{B}v_{s+1}^*, \mathcal{B}e_{s+1}^*\rangle_{\partial\Omega}\big).\end{aligned}$$

By choosing η such that

$$0 < \eta < -\frac{2\left(\langle \mathcal{L}v_{s+1}^*, \mathcal{L}e_{s+1}^*\rangle_\Omega + \lambda^2\langle \mathcal{B}v_{s+1}^*, \mathcal{B}e_{s+1}^*\rangle_{\partial\Omega}\right)}{\left|||v_{s+1}^*|||\right|^2},$$

we would obtain $|||e_s^* - v_{s+1}^\eta|||^2 < |||e_s^* - v_{s+1}^*|||^2$, contradicting the definition of v_{s+1}^* as the minimizer

$$\begin{aligned} v_{s+1}^* = u_{s+1}^* - u_s^* &= \operatorname*{arg\,min}_{v \in \operatorname{span}\Psi_{s+1}} |||u - v||| - u_s^* \\ &= \operatorname*{arg\,min}_{v' \in \operatorname{span}\Psi_{s+1}} |||u - (u_s^* + v')||| + u_s^* - u_s^* \\ &= \operatorname*{arg\,min}_{v' \in \operatorname{span}\Psi_{s+1}} |||e_s^* - v'|||. \end{aligned} \tag{3.3}$$

Therefore, (3.2) must hold. ∎

Proposition 3.1 establishes the fundamental *qualitative* convergence property of Algorithm 2.1. In fact, if we further assume that at the sth stage the trained neural network is sufficiently good to the residual e_s, then we can derive a more delicate

quantitative error estimate regarding the approximation error. To this end, let us consider the space of SLFNs defined by

$$V_N^\sigma = \left\{ f_{NN} : \mathbb{R}^d \to \mathbb{R} \;\middle|\; f_{NN}(\boldsymbol{x}) = \sum_{i=1}^{N} c_i \, \sigma\big(\boldsymbol{w}_i \cdot \boldsymbol{x} + b_i\big) \right\}.$$

The universal approximation property of the SLFN has been established in [12] and is stated as follows.

THEOREM 3.2 (universal approximation [12]). *Suppose that* $1 \le p < \infty, 0 \le s < \infty$ *and that* $\Omega \subset \mathbb{R}^d$ *is compact. If* $\sigma \in C^s(\Omega)$ *is nonconstant and bounded, then the space*

$$V^\sigma := \bigcup_{N=1}^{\infty} V_N^\sigma$$

is dense in the Sobolev space

$$W^{s,p}(\Omega) := \{ v \in L^p(\Omega) : D^\alpha v \in L^p(\Omega) \ \forall |\alpha| \le s \}.$$

In each stage $s > 0$, for any $\tau_s > 0$, the above universal approximation theorem indicates that there exists a network size $n(\tau_s, u_{s-1}^*)$ such that for $n \ge n(\tau_s, u_{s-1}^*)$, one can find $\psi_s \in V_n^\sigma$ satisfying

$$\frac{|||e_{s-1}^* - \psi_s|||}{|||e_{s-1}^*|||} \le \tau_s. \tag{3.4}$$

Assume that sufficient training has been performed such that the relative error bound (3.4) holds. Then we can establish the following proposition for the a posteriori error estimate.

PROPOSITION 3.3. *At stage* $s > 0$, *given* $0 < \tau_s < 1$, *if* (3.4) *holds, then*

$$\frac{1}{1+\tau_s} |||v_s^*||| \le |||u - u_{s-1}^*||| \le \frac{1}{1-\tau_s} |||v_s^*|||. \tag{3.5}$$

Proof. Selecting $v' = \psi_s \in \Psi_s$ in (3.3) and utilizing (3.4), we obtain

$$\big| |||u - u_{s-1}^*||| - |||v_s^*||| \big| \le |||u - u_{s-1}^* - v_s^*||| \le |||e_{s-1}^* - \psi_s||| \le \tau_s |||e_{s-1}^*|||.$$

Thus, we obtain the following bounds:

$$|||u - u_{s-1}^*||| \le |||v_s^*||| + \tau_s |||u - u_{s-1}^*|||$$

and

$$|||u - u_{s-1}^*||| \ge |||v_s^*||| - \tau_s |||u - u_{s-1}^*|||.$$

Combining these inequalities yields (3.5). ∎

Proposition 3.3 demonstrates that $|||v_s|||$ serves as the a posteriori error estimator to guide the stopping criterion. Moreover, the proposed method allows for training a new network to refine the current approximate solution if $|||v_s|||$ exceeds the prescribed tolerance, without requiring the retraining of previous networks.

PROPOSITION 3.4. *Let $S \geq 1$. If $\{\tau_s\}_{s=1}^S$ and $\{n(\tau_s, u_{s-1}^*)\}_{s=1}^S$ are chosen such that (3.4) holds, then*

$$|||u - u_S^*||| \leq |||u - u_0^*||| \cdot \prod_{s=1}^{S} \min\{1, 2\tau_s\}. \tag{3.6}$$

Proof. For each stage $s > 0$, it is obvious that $|||u - u_s^*||| \leq |||u - u_{s-1}^*|||$. In addition, we observe that $u_{s-1}^* + |||u - u_{s-1}^*|||/|||\psi_s||| \cdot \psi_s \in \Psi_s$. According to the definition of u_s^*, we have

$$\begin{aligned} |||u - u_s^*||| &\leq \left|\left|\left| u - \left(u_{s-1}^* + \frac{|||u - u_{s-1}^*|||}{|||\psi_s|||} \cdot \psi_s \right) \right|\right|\right| \\ &= |||u - u_{s-1}^*||| \cdot \left|\left|\left| \frac{u - u_{s-1}^*}{|||u - u_{s-1}^*|||} - \frac{\psi_s}{|||\psi_s|||} \right|\right|\right| \\ &< 2\tau_s |||u - u_{s-1}^*|||. \end{aligned}$$

The final step follows thanks to

$$\begin{aligned} \left|\left|\left| \frac{e_{s-1}^*}{|||e_{s-1}^*|||} - \frac{\psi_s}{|||\psi_s|||} \right|\right|\right| &\leq \left|\left|\left| \frac{e_{s-1}^*}{|||e_{s-1}^*|||} - \frac{\psi_s}{|||e_{s-1}^*|||} \right|\right|\right| + \left|\left|\left| \frac{\psi_s}{|||e_{s-1}^*|||} - \frac{\psi_s}{|||\psi_s|||} \right|\right|\right| \\ &= \frac{|||e_{s-1}^* - \psi_s|||}{|||e_{s-1}^*|||} + \frac{\big||||\psi_s||| - |||e_{s-1}^*|||\big|}{|||e_{s-1}^*|||\,|||\psi_s|||} \cdot |||\psi_s||| \leq 2\tau_s. \end{aligned}$$

Thus, we have

$$|||u - u_s^*||| \leq |||u - u_{s-1}^*||| \cdot \min\{1, 2\tau_s\}.$$

Then the estimate (3.6) follows. ∎

Proposition 3.4 highlights the convergence behavior of Algorithm 2.1. Assuming that (3.4) holds, the algorithm demonstrates favorable convergence characteristics. Furthermore, if $\tau_s = \tau < 1/2$ for all s, then the algorithm achieves geometric convergence.

Remark 3.5. The selection of the network width plays an important role in balancing convergence speed and computational efficiency. Due to its iterative nature, Algorithm 2.1 allows flexibility to vary the network width N_s at each stage. Although in theory a fixed width $N_s \equiv N$ can still achieve convergence by performing sufficient stages [1, Proposition 2.12], it actually results in slow convergence in practice. The primary reason is that a fixed-width network has limited capacity to capture the higher-resolution features of the residual. Proposition 3.4 further suggests how this stagnation can be avoided. Observe that if $\tau_s = \tau < 1/2$ for all s, then the sequence $\{n(\tau, u_{s-1}^*)\}_{s=1}^S$ is expected to be nondecreasing. In particular, the error for u_s^* is reduced by a factor of 2τ compared to the error for u_{s-1}^*. To ensure that ψ_s can capture the higher-resolution features of $u - u_{s-1}^*$ (as compared to the relatively lower-resolution features of $u - u_{s-2}^*$), it is necessary that $n(\tau, u_{s-1}^*) > n(\tau, u_{s-2}^*)$. Therefore, to achieve geometric convergence, the network width N_s must be increased progressively as s grows. In the experiments, following the setting in [1], we set the network width as $N_s = N_0 \times 2^s$, where $N_0 \in \mathbb{Z}^+$ is a predefined width.

4. Numerical results. We apply the proposed solver to several different classes of problems in one and two dimensions, demonstrating the effectiveness of our algorithm. If not mentioned, the prescribed parameters are listed in Appendix A.

For all subsequent examples, if not specified, we set $u_0 = 0$ and use the tanh activation function. The error metrics used in the numerical experiments are the L^∞ and L^2 norms, both of which are approximated on fine equidistant grids. Specifically, a grid of 1000 points is used for the one-dimensional case, while a 300^2 grid is employed for the two-dimensional case. Let the set of these grid points be denoted by X. The errors are then computed as follows:

$$E_{L^\infty} = \max_{\boldsymbol{x}\in X} |u(\boldsymbol{x}) - u_S(\boldsymbol{x})|,$$
$$E_{L^2} = \sqrt{\frac{|\Omega|}{|X|} \sum_{\boldsymbol{x}\in X} |u(\boldsymbol{x}) - u_S(\boldsymbol{x})|^2},$$

where $u(\boldsymbol{x})$ is the exact solution and $u_S(\boldsymbol{x})$ is the numerical solution.

For circular domains, the uniform grid is generated in polar coordinates. The L^∞ error calculation remains unchanged, whereas the L^2 error formula is adjusted to account for the polar grid distribution,

$$E_{L^2} = \sqrt{\frac{P(\Omega)}{|X|} \sum_{\boldsymbol{x}\in X} \|\boldsymbol{x}\| |u(\boldsymbol{x}) - u_S(\boldsymbol{x})|^2},$$

where $P(\Omega)$ denotes the perimeter of the circular domain.

4.1. Function fitting. We begin by considering a function fitting problem. The target function f is composed of the first four terms of the Fourier series for a square wave:

$$f(x) = \sin(x) + \frac{1}{3}\sin(3\pi x) + \frac{1}{5}\sin(5\pi x) + \frac{1}{7}\sin(7\pi x).$$

We present the true and a posteriori errors of stages 2, 4, and 6, along with the convergence result of L^2 and L^∞ errors with respect to the number of stages in Figure 2. Initially, the low-frequency components of the error are learned, with subsequent stages focusing on the high-frequency error components. The error curves demonstrate that the proposed method has exponential convergence, achieving an L^∞ error of 1.689e-11 and an L^2 error of 3.652e-12 after seven stages. In the right panel of Figure 2, we also present results from GNN [1] using its public code. For fairness, we adopt the same network width and keep other settings unchanged. Our method achieves significantly higher accuracy and is markedly faster, requiring only 540 ms on a CPU compared to GNN's 57.7 s. We report CPU runtimes since the GNN code does not fully support GPU acceleration (e.g., matrix assembly remains CPU-bound), while our implementation supports both CPU and GPU.

4.2. Advection-diffusion problem with boundary layer. We consider the following one-dimensional convection-diffusion equation with a boundary layer:

$$-\varepsilon u'' + bu' = f, \quad x \in \Omega = (-1, 1),$$
$$u = 0, \quad x \in \partial\Omega.$$

The solution to this problem consists of a smooth component and a boundary layer term of the form $e^{\frac{b}{\varepsilon}(x-1)}$. For instance, when $f \equiv 1$, the exact solution is given by

$$u(x) = \frac{x}{b} + \frac{1}{b}\left(\frac{e^{-\frac{2b}{\varepsilon}} + 1}{e^{-\frac{2b}{\varepsilon}} - 1}\right)\left(\frac{2e^{\frac{b}{\varepsilon}(x-1)}}{e^{-\frac{2b}{\varepsilon}} + 1} - 1\right).$$

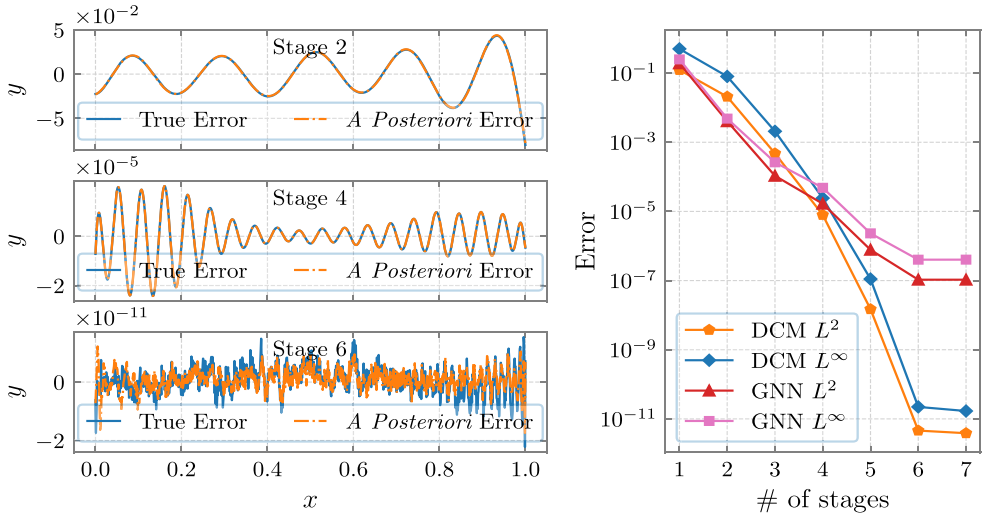

FIG. 2. *Results for subsection* 4.1. *Left: The true and a posteriori errors of stages* 2, 4, *and* 6. *Right: Error versus number of stages.*

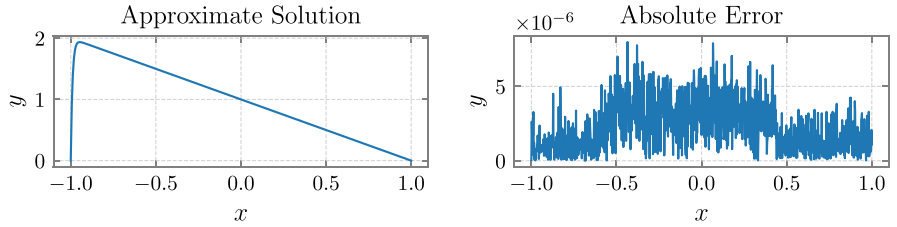

FIG. 3. *Approximation solution and the corresponding absolute error for subsection* 4.2.

This example focuses on the difficulty in accurately resolving the boundary layer, particularly for small values of ε. For numerical testing, we set $\varepsilon = 10^{-2}$ and $b = -1$.

Figure 3 presents the approximate solution and the absolute error for the final stage. Figure 4 shows the training result of stages $s = 2, 3, 4$ and L^2, L^∞ errors at each stage of the iteration. From the error curves, we observe that the boundary layer is initially not well captured. However, with successive corrections, our method successfully captures the boundary layer with high accuracy. As in subsection 4.1, DCM again outperforms GNN in both accuracy and efficiency: The CPU runtimes are 14 s for DCM and 109 s for GNN.

4.3. Poisson equation. The previous examples demonstrate the effectiveness of the algorithm. This subsection aims to show that the proposed method serves as a general framework which can be further enhanced by integrating with other strategies to improve the solution of some challenging problems. Consider the following Poisson equation:

$$\begin{aligned} -\Delta u &= f \qquad \text{in } \Omega, \\ u &= g \qquad \text{on } \partial\Omega. \end{aligned} \tag{4.1}$$

In this subsection, we analyze two challenging cases for solving (4.1): The first examines an L-shaped domain where geometric singularities lead to nonsmooth

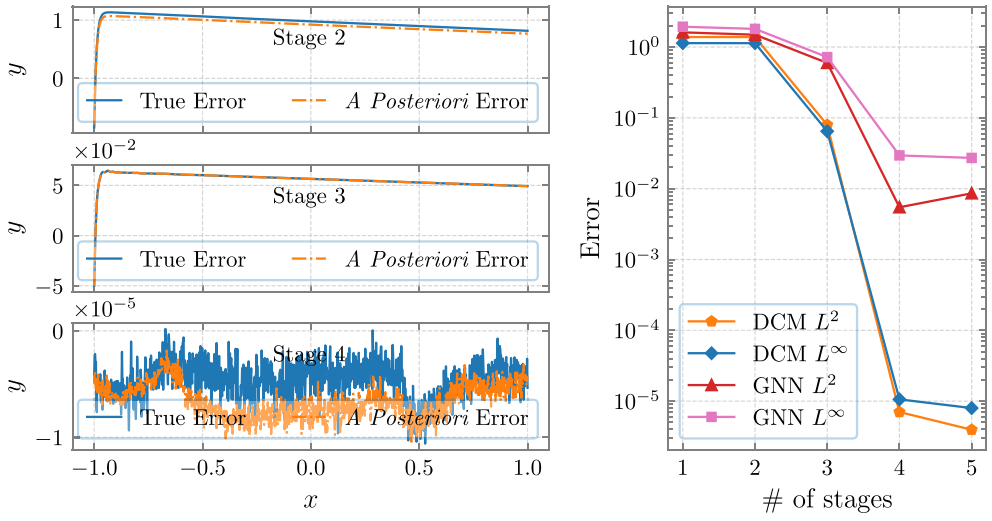

FIG. 4. *Results for subsection* 4.2. *Left: The true and a posteriori errors of stages* 2, 3, *and* 4. *Right: Error versus number of stages.*

solutions, and the second focuses on a two-dimensional problem where the source term exhibits rapid, localized variations, making it challenging to resolve near these regions.

4.3.1. L-shaped domain. We consider the Poisson equation (4.1) on an L-shaped domain $\Omega = (-1,1)^2 \setminus (-1,0]^2$ with a constant source term $f \equiv 1$ and homogeneous Dirichlet boundary condition $g \equiv 0$. The primary challenge in this problem arises from the presence of a nonconvex corner at the origin where the solution exhibits singular behavior.

As established in [3], the solution of the Poisson equation in an L-shaped domain can be expressed as a series expansion that accounts for both singular and smooth components. Specifically, the solution is given by

$$u(x,y) = \sum_{i=1}^{\infty} c_i r^{\lambda_i} \sin(\lambda_i \theta) + u^*(x,y), \quad \lambda_i = \frac{2}{3} i,$$

where $u^* \in H^2(\Omega)$ represents the smooth part of the solution. The singular part, characterized by terms of the form $r^{\lambda_i} \sin(\lambda_i \theta)$, captures the behavior near the nonconvex corner.

In this experiment, we enhance the network by incorporating knowledge-based hidden neurons that correspond to the singular terms $r^{\lambda_i} \sin(\lambda_i \theta)$ for i up to N_k in each training stage. This approach leverages the known structure of the solution to improve accuracy.

Figure 5 illustrates the absolute errors obtained without (left) and with (middle) the inclusion of knowledge-based hidden neurons. It can be observed that incorporating these neurons leads to improved accuracy, with a more consistent decay in the error curve. Notably, the knowledge-based hidden neurons do not introduce additional parameters for optimization; instead, they only slightly increase the size of the least-squares system constructed by CollocLSQ. Consequently, the improvement in accuracy is achieved with a minimal increase in computational cost. Figure 6

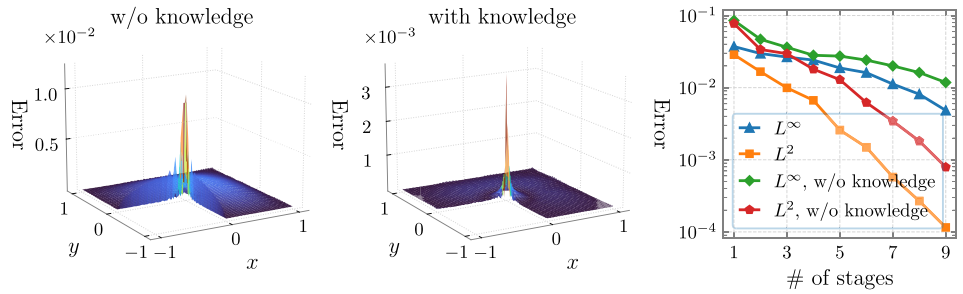

FIG. 5. *Poisson equation on an L-shaped domain. The absolute errors obtained without (left) and with (middle) the inclusion of knowledge-based hidden neurons. Right: L^2 and L^∞ errors computed at each stage.*

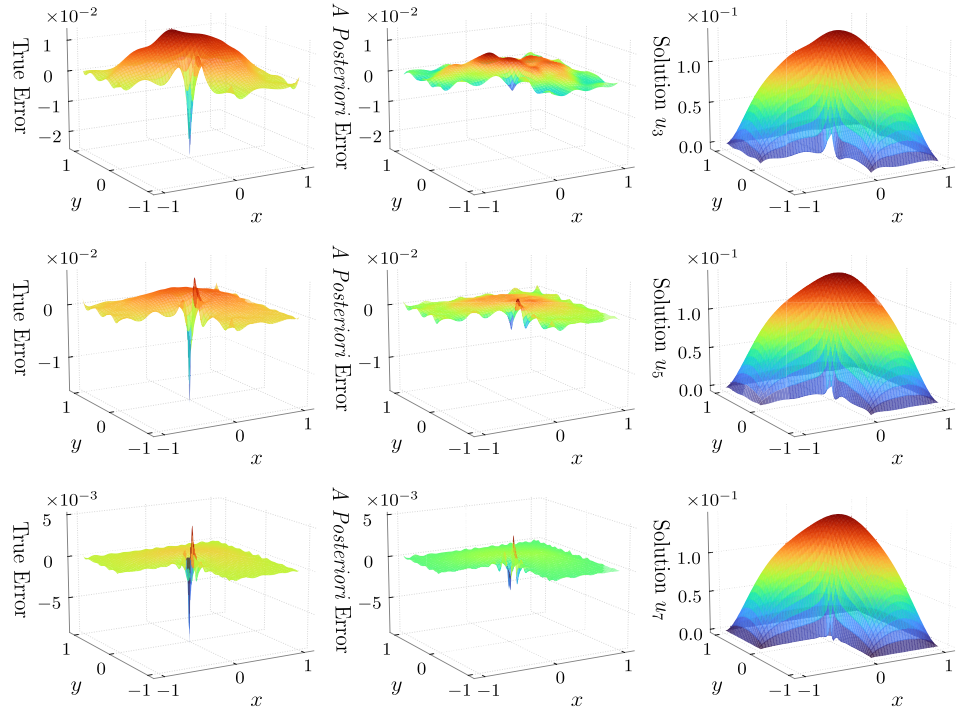

FIG. 6. *Poisson equation on an L-shaped domain with knowledge-based hidden neurons. True errors (left column), a posteriori errors (middle column), and numerical solution u_s (right column) of stages 3, 5, and 7.*

presents the training results obtained using knowledge-based hidden neurons at stages $s = 3, 5, 7$, which include the true errors (left column), the a posteriori errors (middle column), and numerical solutions (right column).

4.3.2. Rapidly varying source term. We study the Poisson equation (4.1) over the domain $\Omega = (0,1)^2$ with the exact solution given by

$$u(\boldsymbol{x}) = 16\,(1-x_1)\,x_1\,(1-x_2)\,x_2\left(\frac{1}{2}+\frac{1}{\pi}\arctan\left(\frac{r-\|\boldsymbol{x}-\boldsymbol{c}\|^2}{\varepsilon}\right)\right), \tag{4.2}$$

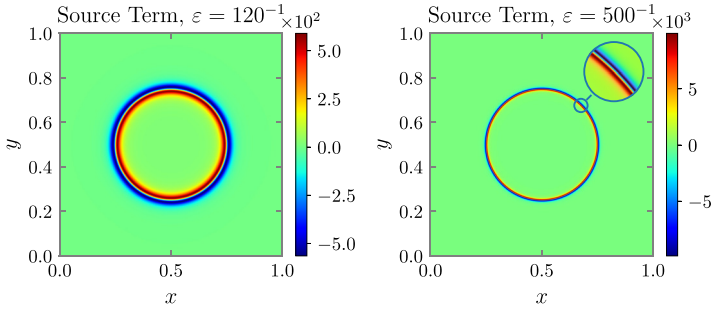

FIG. 7. *Poisson equation with a rapidly varying source term. The source term corresponds to* $\varepsilon = 120^{-1}$ *(left) and* $\varepsilon = 500^{-1}$ *(right), with a zoomed-in view.*

where $r = 1/16$ and $\boldsymbol{c} = (1/2, 1/2)$, with the parameter ε controlling the sharpness of the transition region in the solution. We consider two values of ε: $1/120$ and $1/500$. Smaller values of ε result in sharper gradients and more localized variations near the boundary of a ball centered at $\boldsymbol{c}$ with radius $\sqrt{r}$. The corresponding source terms for these two cases are shown in Figure 7. Capturing such steep and localized variations in the source term is a challenging task.

To address this issue, in addition to the (global) neuron functions using tanh as the activate function, we introduce a kind of localized neuron function, inspired by the layer growth technique given in [7]. Given an approximate solution u^* and a corresponding set of points $\{\boldsymbol{x}_i\}_{i=1}^{N_L}$, we construct a new network with N_L localized hidden neurons defined by

$$\phi_i(\boldsymbol{x}) := \exp\left(-\|\boldsymbol{k}_i \odot (\boldsymbol{x} - \boldsymbol{x}_i)\|^2\right) \exp\left(-\frac{1}{2}\|\boldsymbol{k}_i\|^2 \left(u^*(\boldsymbol{x}) - u^*(\boldsymbol{x}_i)\right)^2\right)$$

for $i = 1, \ldots, N_L$, where $\boldsymbol{k}_i \sim \mathbb{U}(-R_L, R_L)$ is a shape parameter that modulates the localization effect and $\odot$ denotes elementwise multiplication. The localized neuron function $\phi_i(\boldsymbol{x})$ is highly concentrated around the level set $\{\boldsymbol{x} : u^*(\boldsymbol{x}) = u^*(\boldsymbol{x}_i)\}$ and is further confined by a Gaussian function centered at $\boldsymbol{x}_i$. In this work, we determine the points $\{\boldsymbol{x}_i\}_{i=1}^{N_L}$ using Algorithm 2.3, with the sampling distribution function being the residual of the PDE, $f - \mathcal{L}u^*$. We call these neurons adaptive localized neurons. After initialization, the network is trained using the CollocLSQ method without further optimization steps.

In contrast to the adaptive initialization strategy discussed in subsection 2.3.2, which focuses on capturing different frequency components and approximating the error distribution through globally defined neuron functions, the adaptive localized neuron approach constructs neuron functions that are highly localized around specific points, similar to radial basis functions. This localization allows the method to effectively capture the sharp and localized features of the source term.

The success of the adaptive approach hinges on the quality of the initial approximation u^*. As such, the proposed method using tanh as the activate function is first applied iteratively over S_1 stages to obtain a sufficiently accurate initial solution. After that, the adaptive localized neuron strategy is employed over an additional S_2 stages to further refine the solution. For consistency, we continue to denote the total number of stages by S, i.e., $S = S_1 + S_2$.

The training results for both cases are presented in Figure 8. For $\varepsilon = 120^{-1}$, the method achieves an L^∞ error of 9.924e-7 and an L^2 error of 1.969e-7, while for

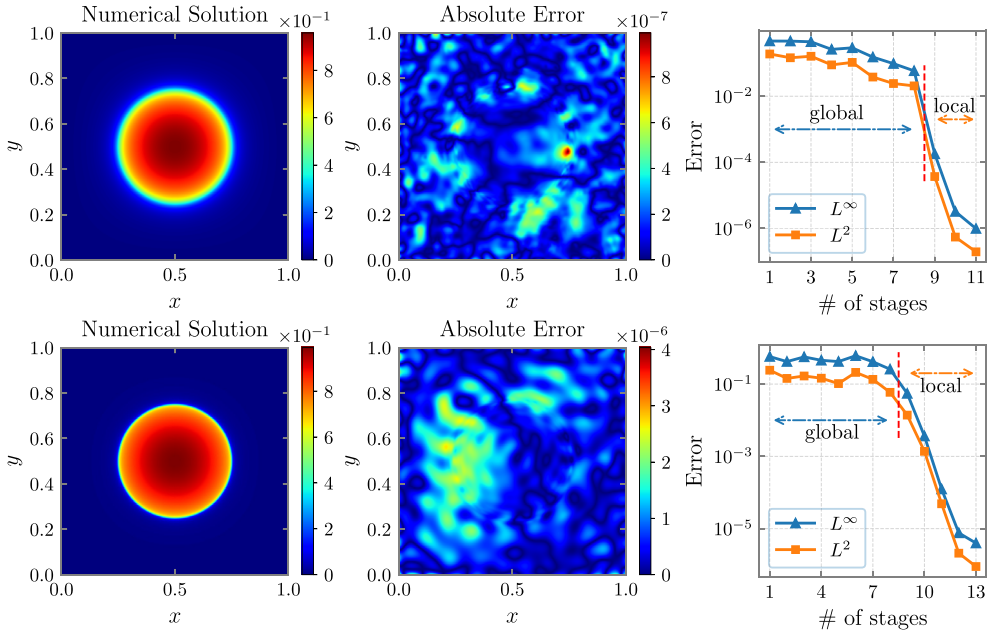

FIG. 8. *Poisson equation with a rapidly varying source term. Final-stage solution and error, with L^2 and L^∞ errors at each stage. Top row: $\varepsilon = 120^{-1}$; bottom row: $\varepsilon = 500^{-1}$.*

$\varepsilon = 500^{-1}$, the corresponding errors are 3.978e-6 and 8.951e-7. The third column of Figure 8 plots the error curves versus the training stages. We annotate the initial stages using the baseline deep collocation method as "global" and the subsequent refinement stages employing localized neuron functions as "local." It is noteworthy that as ε decreases (i.e., as the source term becomes more localized), the error curve exhibits slower convergence and increased oscillations, indicating the greater difficulty in resolving sharper transitions. Nevertheless, the introduction of the "local" refinement stages significantly reduces the errors in both cases to the same small scale. This underscores the effectiveness of the localization strategy in achieving high accuracy for problems with sharply localized features.

4.4. Fourth-order problem: Biharmonic equation. In this subsection, we evaluate the proposed method for the biharmonic equation, a fourth-order PDE, in both smooth and singular cases.

4.4.1. Smooth solution. Consider the biharmonic equation given by

$$\begin{aligned} \Delta^2 u &= f, && \boldsymbol{x} \in \Omega, \\ u &= \partial_n u = 0, && \boldsymbol{x} \in \partial\Omega. \end{aligned}$$

We test this on the domain $\Omega = (-1,1)^2$ with a smooth exact solution

$$u(\boldsymbol{x}) = \sin^2(\pi x)\sin^2(\pi y) + \left(1-x^2\right)^4\left(1-y^2\right)^4.$$

The training results are shown in Figure 9. The error curve demonstrates that the proposed method effectively handles the smooth case, with the error decreasing exponentially and achieving an L^∞ error of 1.399e-4 and an L^2 error of 5.526e-5 after seven stages.

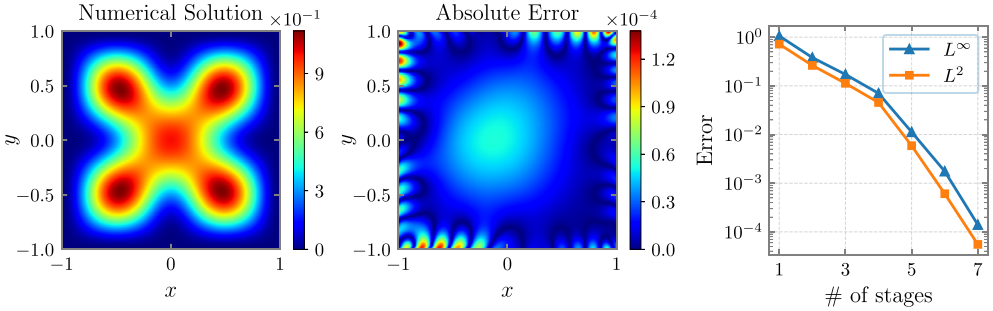

FIG. 9. *Biharmonic equation with smooth solution. Final-stage approximate solution and error, with L^2 and L^∞ errors computed at each stage.*

4.4.2. Unit circle with point load. We consider the biharmonic equation with a point load source term in the unit disk $\Omega = \{(x, y) : x^2 + y^2 < 1\}$:

$$\Delta^2 u = \delta_{\{(0,0)\}} \quad \text{in } \Omega, \tag{4.3a}$$

$$u - \varepsilon_1 \partial_n(\Delta u) = 0 \quad \text{on } \partial\Omega, \tag{4.3b}$$

$$\Delta u + \varepsilon_2 \partial_n u = 0 \quad \text{on } \partial\Omega. \tag{4.3c}$$

The Dirac delta function is defined in the sense of distributions. To avoid explicitly computing the Dirac function, we replace (4.3a) with the following equivalent governing equation:

$$2\pi r(\Delta u)_r = 1 \qquad \text{in } \Omega.$$

Although the source term is no longer singular and the equation becomes third order, the solution remains challenging to solve due to its low regularity, with a singular behavior in the second derivative at the origin, given by

$$u(r,\theta) = \frac{r^2}{8\pi} \ln r + c_1 r^2 + c_2, \quad c_1 = \frac{1}{4 + 2\varepsilon_2} \left(-\frac{1}{2\pi} - \frac{\varepsilon_2}{8\pi} \right), \quad c_2 = -c_1 + \frac{\varepsilon_1}{2\pi}.$$

Figure 10 shows the training results for stages $s = 1, 3, 5$, including the true errors (left column), the a posteriori errors (middle column), and the approximate solutions (right column). From Figure 11, we observe that the logarithmic singularity at the origin slows down the error reduction; however, relatively good accuracy is still achieved, with an L^∞ error of 1.028e-4 and an L^2 error of 3.304e-5.

4.5. Nonlinear problem: Allen–Cahn equation. Note that the GNN is very time consuming since it needs to compute the inner products with high accuracy. This prevents GNN from being used for nonlinear or time-dependent problems. Instead, the present collocation-type method is more efficient in solving nonlinear or time-dependent problems. To this end, we consider the following nonlinear steady-state Allen–Cahn equation on $\Omega = (-1, 1)^2$:

$$-\varepsilon^2 \Delta u + u^3 - u = 0 \qquad \text{in } \Omega,$$
$$u = 0 \qquad \text{on } \partial\Omega.$$

A fixed-point iteration method is employed to handle this nonlinear equation, with the following equation being solved at each iteration:

$$\varepsilon^2 \Delta u_k - \alpha\varepsilon^2 u_k = u_{k-1}^3 - \left(1 + \alpha\varepsilon^2\right) u_{k-1}, \tag{4.4}$$

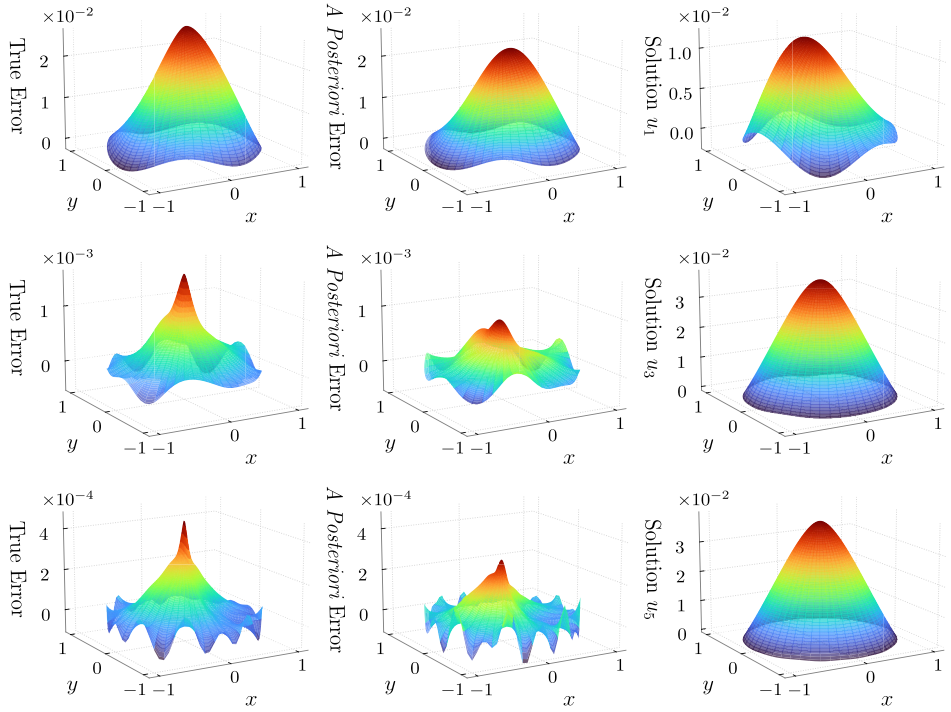

FIG. 10. *Biharmonic equation with point load. True errors (left column) and a posteriori errors (middle column) as well as numerical solutions u_s (right column) for $s = 1, 3, 5$.*

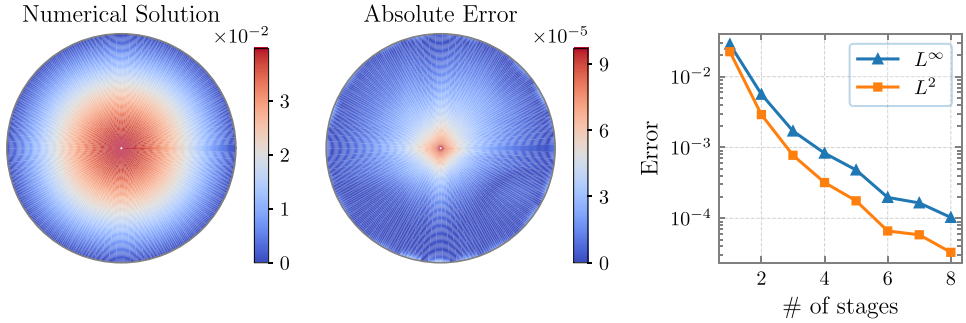

FIG. 11. *Biharmonic equation with point load. Final-stage approximate solution and error, with L^2 and L^∞ errors computed at each stage.*

where α is a constant introduced to enhance iteration stability. Denote the linearized operator by $\mathcal{L} = \varepsilon^2\Delta - \alpha\varepsilon^2$ and the source term at each iteration as $f(u_{k-1}) = u_{k-1}^3 - (1+\alpha\varepsilon^2)u_{k-1}$.

Suppose that after stage s, we have an approximation $u_s \in \text{span}\Psi_s$, where $\Psi_s = \{\psi_1, \ldots, \psi_s\}$ denotes the set of basis functions. The procedure for constructing the solution at stage $s+1$ is described below:

(1) Initialize with $i = 0$, set $\tilde{u}_{s+1}^0 = u_s$, and construct a new SLFN basis ψ_{s+1}^{i+1} using adaptive initialization (Algorithm 2.4) based on the residual $f^{\text{res}} = f(u_s) - \mathcal{L}u_s$.

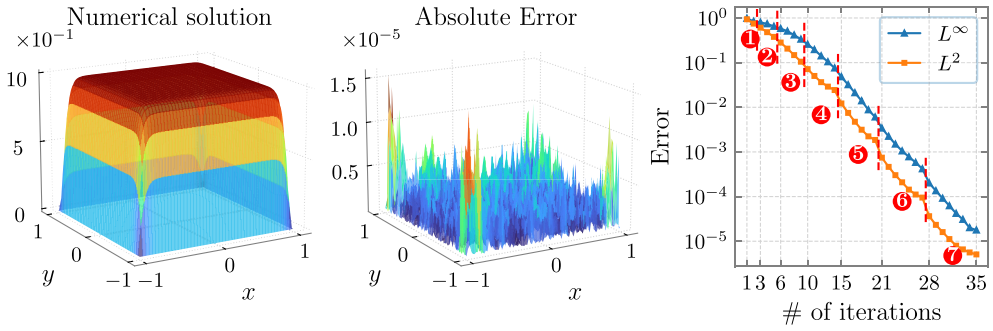

FIG. 12. *Steady-state Allen–Cahn equation. Final-stage approximate solution and error, with L^2 and L^∞ errors computed at each nonlinear iteration.*

(2) Generate new residual-based internal collocations using rejection sampling (Algorithm 2.3).
(3) Solve (4.4) for the approximation $u_s^i \in \text{span}\Psi_s$ using CollocLSQ with the source term $f(\tilde{u}_{s+1}^i)$.
(4) Train the new basis function ψ_{s+1}^{i+1} to approximate the residual equation of (4.4) by Algorithm 2.2 with $f^{\text{res}} = f(\tilde{u}_{s+1}^i) - \mathcal{L}u_s^i$ and $g^{\text{res}} = -u_s^i$.
(5) Update the solution $\tilde{u}_{s+1}^{i+1} \in \text{span}(\Psi_s \cup \{\psi_{s+1}^{i+1}\})$ using CollocLSQ with source term $f(\tilde{u}_{s+1}^i)$.
(6) Set $i \leftarrow i+1$, and repeat steps (2)–(6) until a predefined stopping criterion is met.
(7) Suppose that the process terminates after I_{s+1} iterations, then update the solution and basis set as

$$u_{s+1} = \tilde{u}_{s+1}^{I_{s+1}}, \quad \Psi_{s+1} = \Psi_s \cup \{\psi_{s+1}^{I_{s+1}}\}.$$

Regarding the stopping criterion in step 6, for simplicity in the experiments, we predefine the number of nonlinear iterations for each stage to prevent excessive iterations from causing error saturation. Of course, more adaptive criteria could be designed to better control the iterative process.

For the numerical experiment, we set $\varepsilon = 0.05$ and $\alpha\varepsilon^2 = 10$ and use the following initial guess:

$$u_0(r) = \begin{cases} 1 & \text{if } r \le R_0, \\ \frac{\log(R_1/r)}{\log(R_1/R_0)} & \text{if } R_0 < r < R_1, \\ 0 & \text{if } r \ge R_1, \end{cases}$$

where $R_0 = 0.7$, $R_1 = 0.9$.

Figure 12 shows the approximate solution and error at the final stage, along with the L^2 and L^∞ errors at each nonlinear iteration. The error curves clearly indicate that the proposed method progressively improves the solution accuracy, achieving an L^∞ error of 1.775e-5 and an L^2 error of 5.067e-6 after seven stages and a total of 35 nonlinear iterations. Figure 13 further illustrates the true errors (left column), the a posteriori errors (middle column), and the numerical solutions u_s (right column) for stages $s = 2, 4, 6$.

4.6. Three-dimensional problem: Poisson equation. For the final example, we compare the performance of our proposed method with the random feature model and TransNet [33]. As discussed in subsection 2.1, neural network–based methods involve numerous hyperparameters, making it challenging to compare

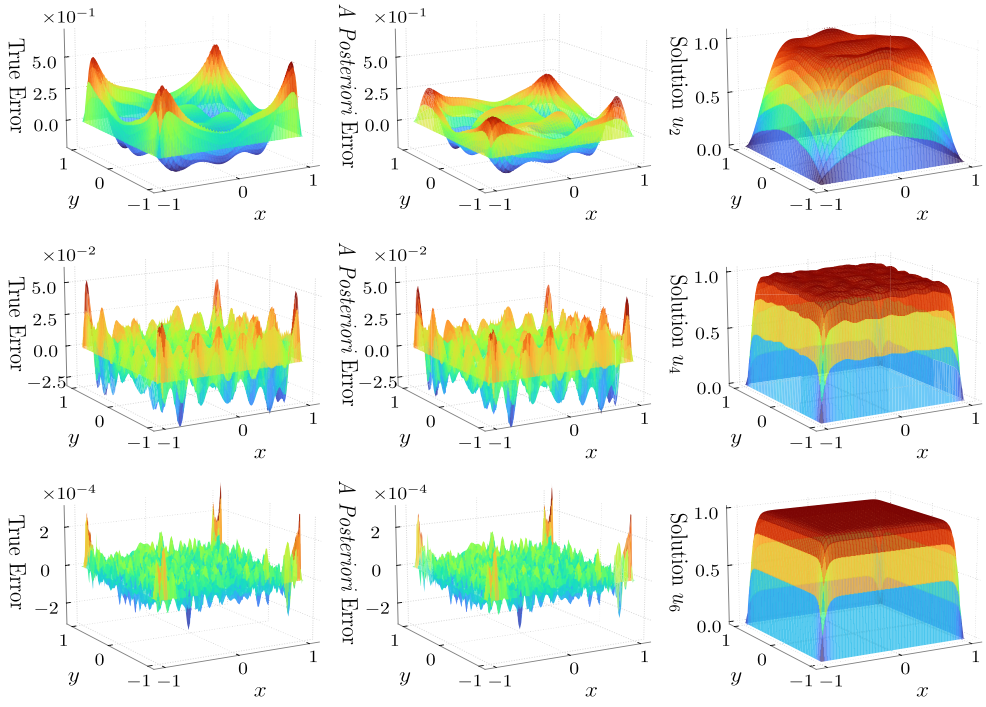

FIG. 13. *Steady-state Allen–Cahn equation. True errors (left column) and a posteriori errors (middle column) as well as numerical solutions u_s (right column) for $s = 2, 4, 6$.*

different approaches in a fair and consistent manner. To address this, we adopt the benchmark problem (C6) from [33], which involves solving a three-dimensional Poisson equation (4.1) on the domain $\Omega = [-1, 1]^3$ with the test solution $u(\boldsymbol{x}) = \sin(kx_1)\sin(kx_2)\sin(kx_3)$, where $k = 2\pi$. Thanks to Zhang et al. [33], we are able to reproduce their results using their publicly available code and well-tuned hyperparameters. We only modified the code to compute the L^2 and L^∞ errors, as the original implementation only reported the mean squared error. It should be noted that the random feature model used here is exactly the one in [33], which corresponds to the RFM [6] without domain decomposition.

Figure 14 presents the L^2 and L^∞ errors versus iteration number for the proposed deep collocation method, the random feature model, and TransNet. For both the random feature model and TransNet, the result at iteration s is obtained using a neural network with width 80×2^s, which matches the network width employed by the deep collocation method at the same iteration. This setup ensures a fair comparison in terms of model capacity. It is evident that the deep collocation method consistently outperforms the other two methods by one to two orders of magnitude across all iterations. The superiority is especially notable in the L^∞ norm due to the residual-driven adaptive sampling, which effectively targets regions with large pointwise errors.

5. Conclusions. Deep learning methods have demonstrated great promise in solving PDEs. However, they suffer from accuracy and high computational costs. In this paper, we developed an efficient and high-accuracy neural network–based approach with robust training for solving PDEs. In particular, we proposed an adaptive method by iteratively constructing a sequence of basis functions using single-

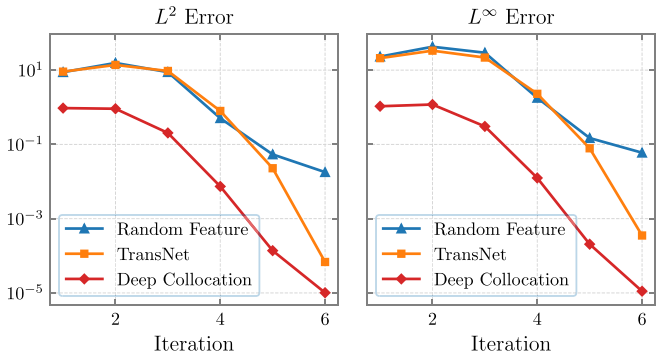

FIG. 14. *Comparison of the deep collocation method, the random feature model, and TransNet for the three-dimensional Poisson example.*

hidden-layer neural networks, which can be trained efficiently through the proposed training strategy. Additionally, we developed adaptive strategies for network initialization and collocation sampling, which not only addressed common training challenges but also simplified the tuning of hyperparameters, ensuring robust and accurate performance across a wide range of settings.

We provided an analysis showing that the accuracy of the method improved progressively with each iteration. Furthermore, we derived a reliable a posteriori error estimator and provided theoretical insights for achieving geometric convergence. We demonstrated both the accuracy and the stability of the proposed approach by conducting numerical experiments on various problems, including function fitting, boundary layer problems, the Poisson equation, the biharmonic equation, and the Allen–Cahn equation. Additionally, the a posteriori error estimator proved effective in all cases.

The proposed method excels not only in convergence and robustness but also in its broad applicability. Thanks to its collocation-based nature, the algorithm is well suited to a wide variety of problems, including those involving nonlinearity and complex geometries. Moreover, it can be easily extended to incorporate advanced strategies in tackling specific challenges, further enhancing both accuracy and efficiency, as evidenced by the numerical examples for the Poisson equation. Future work will focus on refining the adaptive basis construction technique and exploring its potential applications to even more complex and higher-dimensional problems.

Appendix A. Parameters for numerical examples. This section outlines the details and notations used in the numerical experiments:

- In all experiments, we set the initial condition as $u_0 = 0$ and use tanh as the activation function.
- Candidate points for rejection sampling are randomly drawn from a fine grid, consisting of 1000 points per dimension. This ensures that the selected sample points are well distributed and not overly clustered.

The following notations are used throughout the experiments:

- M_Ω^{I}, M_Ω^{II}, and $M_{\partial\Omega}$ represent the number of uniform and adaptive collocation points within the domain and the number of boundary points, respectively.
- λ is the regularization parameter in the loss function.
- S denotes the total number of stages in the algorithm.

TABLE 1
Parameters used in the numerical experiments.

Examples	M_Ω^{I}	M_Ω^{II}	$M_{\partial\Omega}$	λ	S	N_s	R_s	N_{opt}	α
Section 4.1	512	256	2	1	6	$5\times 2^{s-1}$	$3s-2$	10	5e-2
Section 4.2	512	256	2	10	5	20×2^{s}	$3s+7$	20	5e-2
Section 4.3.1	1e4	5e3	4e3	1e3	9	10×2^{s}	$2s-1$	10	5e-3
Section 4.3.2 $\varepsilon=120^{-1}$	1e4	5e3	4e3	1e3	11	10×2^{s}	$3s-2$	10	5e-2
Section 4.3.2 $\varepsilon=500^{-1}$	9e4	4.5e4	1.6e4	1e3	13	10×2^{s}	$3s-2$	10	5e-2
Section 4.4.1	1e4	5e3	4e3	1e3	7	20×2^{s}	$s+1$	10	5e-3
Section 4.4.2	1e4	5e3	4e3	1e3	8	10×2^{s}	s	10	5e-3
Section 4.5	1e4	5e3	4e2	1e3	7	20×2^{s}	s	10	5e-3
Section 4.6	8e3	4e3	2.4e3	1e3	6	80×2^{s}	1	10	5e-3

TABLE 2
Parameters for the adaptive localized neuron strategy used in section 4.3.2.

Cases	S_1	S_2	N_L	R_L
$\varepsilon=120^{-1}$	8	3	1500	10
$\varepsilon=500^{-1}$	8	5	1500	10

- N_s represents the number of hidden neurons in the SLFN at stage s, where $s=1,2,\ldots,S$.
- R_s is the scaling factor for the weights at stage s, where $s=1,2,\ldots,S$.
- N_{opt} is the maximum number of optimization steps in the Adam optimizer.
- α denotes the learning rate in the Adam optimizer.

The detailed parameters used in the numerical experiments are presented in Table 1. In section 4.3.1, which examines the Poisson equation on an L-shaped domain, the knowledge-based test incorporates singular terms of the form $r^{\lambda_i}\sin(\lambda_i\theta)$ for i up to $N_k=20$ in each training stage. All other parameters remain consistent with those used in the standard case, as listed in Table 1. Table 2 provides additional parameters for localized training used in section Section 4.3.2.

In section 4.5, which considers the steady-state Allen–Cahn equation, the number of iterations per stage is set as $I_s=s+1$.

REFERENCES

[1] M. AINSWORTH AND J. DONG, *Galerkin neural networks: A framework for approximating variational equations with error control*, SIAM J. Sci. Comput., 43 (2021), pp. A2474–A2501, https://doi.org/10.1137/20M1366587.

[2] M. AINSWORTH AND J. DONG, *Galerkin neural network approximation of singularly-perturbed elliptic systems*, Comput. Methods Appl. Mech. Engrg., 402 (2022), 115169, https://doi.org/10.1016/j.cma.2022.115169.

[3] M. AINSWORTH AND J. DONG, *Extended Galerkin neural network approximation of singular variational problems with error control*, SIAM J. Sci. Comput., 47 (2025), pp. C738–C768.

[4] Z. ALDIRANY, R. COTTEREAU, M. LAFOREST, AND S. PRUDHOMME, *Multi-level neural networks for accurate solutions of boundary-value problems*, Comput. Methods Appl. Mech. Engrg., 419 (2023), 116666, https://doi.org/10.1016/j.cma.2023.116666.

[5] G. CASELLA, C. P. ROBERT, AND M. T. WELLS, *Generalized accept-reject sampling schemes*, IMS Lecture Notes Monogr. Ser., (2004), pp. 342–347, https://doi.org/10.1214/lnms/1196285403.

[6] J. Chen, X. Chi, W. E, and Z. Yang, *Bridging traditional and machine learning-based algorithms for solving PDEs: The random feature method*, J. Mach. Learn., 1 (2022), pp. 268–298, https://doi.org/10.4208/jml.220726.

[7] H. Dang and F. Wang, *Adaptive Growing Randomized Neural Networks for Solving Partial Differential Equations*, preprint, arXiv:2408.17225, 2024.

[8] S. Dong and Z. Li, *Local extreme learning machines and domain decomposition for solving linear and nonlinear partial differential equations*, Comput. Methods Appl. Mech. Engrg., 387 (2021), 114129, https://doi.org/10.1016/j.cma.2021.114129.

[9] G. Fabiani, F. Calabrò, L. Russo, and C. Siettos, *Numerical solution and bifurcation analysis of nonlinear partial differential equations with extreme learning machines*, J. Sci. Comput., 89 (2021), 44, https://doi.org/10.1007/s10915-021-01650-5.

[10] Z. Gao, L. Yan, and T. Zhou, *Failure-informed adaptive sampling for PINNs*, SIAM J. Sci. Comput., 45 (2022), pp. 1971–1994, https://doi.org/10.1137/22M1527763.

[11] J. Han, A. Jentzen, and W. E, *Solving high-dimensional partial differential equations using deep learning*, Proc. Natl. Acad. Sci., 115 (2018), pp. 8505–8510, https://doi.org/10.1073/pnas.1718942115.

[12] K. Hornik, *Approximation capabilities of multilayer feedforward networks*, Neural Netw., 4 (1991), pp. 251–257, https://doi.org/10.1016/0893-6080(91)90009-T.

[13] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, *Extreme learning machine: Theory and applications*, Neurocomputing, 70 (2006), pp. 489–501, https://doi.org/10.1016/j.neucom.2005.12.126.

[14] B. Igelnik and Y.-H. Pao, *Stochastic choice of basis functions in adaptive function approximation and the functional-link net*, IEEE Trans. Neural Netw., 6 (1995), pp. 1320–1329, https://doi.org/10.1109/72.471375.

[15] W. T. Leung, G. Lin, and Z. Zhang, *NH-PINN: Neural homogenization-based physics-informed neural network for multiscale problems*, J. Comput. Phys., 470 (2022), 111539, https://doi.org/10.1016/j.jcp.2022.111539.

[16] L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, and S. G. Johnson, *Physics-informed neural networks with hard constraints for inverse design*, SIAM J. Sci. Comput., 43 (2021), pp. B1105–B1132, https://doi.org/10.1137/21M1397908.

[17] Z. Mao, A. D. Jagtap, and G. E. Karniadakis, *Physics-informed neural networks for high-speed flows*, Comput. Methods Appl. Mech. Engrg., 360 (2020), 112789, https://doi.org/10.1016/j.cma.2019.112789.

[18] Z. Mao and X. Meng, *Physics-informed neural networks with residual/gradient-based adaptive sampling methods for solving partial differential equations with sharp solutions*, Appl. Math. Mech., 44 (2023), pp. 1069–1084, https://doi.org/10.1007/s10483-023-2994-7.

[19] Y.-H. Pao, G.-H. Park, and D. J. Sobajic, *Learning and generalization characteristics of the random vector functional-link net*, Neurocomputing, 6 (1994), pp. 163–180, https://doi.org/10.1016/0925-2312(94)90053-1.

[20] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville, *On the spectral bias of neural networks*, in Proceedings of the 36th International Conference on Machine Learning, PMLR, 2019, pp. 5301–5310.

[21] M. Raissi, P. Perdikaris, and 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), pp. 686–707, https://doi.org/10.1016/j.jcp.2018.10.045.

[22] J. Sirignano and K. Spiliopoulos, *DGM: A deep learning algorithm for solving partial differential equations*, J. Comput. Phys., 375 (2018), pp. 1339–1364, https://doi.org/10.1016/j.jcp.2018.08.029.

[23] K. Tang, X. Wan, and C. Yang, *DAS-PINNs: A deep adaptive sampling method for solving high-dimensional partial differential equations*, J. Comput. Phys., 476 (2023), 111868, https://doi.org/10.1016/j.jcp.2022.111868.

[24] S. Wang, A. K. Bhartari, B. Li, and P. Perdikaris, *Gradient Alignment in Physics-Informed Neural Networks: A Second-Order Optimization Perspective*, preprint, arXiv:2502.00604, 2025.

[25] Y. Wang and S. Dong, *An extreme learning machine-based method for computational PDEs in higher dimensions*, Comput. Methods Appl. Mech. Engrg., 418 (2024), 116578, https://doi.org/10.1016/j.cma.2023.116578.

[26] Y. Wang and C.-Y. Lai, *Multi-stage neural networks: Function approximator of machine precision*, J. Comput. Phys., 504 (2024), 112865, https://doi.org/10.1016/j.jcp.2024.112865.

[27] E. Weinan and B. Yu, *The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems*, Commun. Math. Stat., 6 (2018), pp. 1–12.

[28] Y. Xu, *Multi-grade deep learning*, Commun. Appl. Math. Comput., (2025), pp. 1–52.
[29] Y. Xu and T. Zeng, *Multi-Grade Deep Learning for Partial Differential Equations with Applications to the Burgers Equation*, preprint, arXiv:2309.07401, 2023.
[30] Z.-Q. J. Xu, Y. Zhang, T. Luo, Y. Xiao, and Z. Ma, *Frequency principle: Fourier analysis sheds light on deep neural networks*, Commun. Comput. Phys., 28 (2020), pp. 1746–1767, https://doi.org/10.4208/cicp.OA-2020-0085.
[31] Y. Yang, M. Hou, and J. Luo, *A novel improved extreme learning machine algorithm in solving ordinary differential equations by Legendre neural network methods*, Adv. Differential Equations, 2018 (2018), 469, https://doi.org/10.1186/s13662-018-1927-x.
[32] Y. Zang, G. Bao, X. Ye, and H. Zhou, *Weak adversarial networks for high-dimensional partial differential equations*, J. Comput. Phys., 411 (2019), 109409, https://doi.org/10.1016/j.jcp.2020.109409.
[33] Z. Zhang, F. Bao, L. Ju, and G. Zhang, *Transferable neural networks for partial differential equations*, J. Sci. Comput., 99 (2024), 2, https://doi.org/10.1007/s10915-024-02463-y.
[34] Z. Zhang, J. Li, and B. Liu, *Annealed adaptive importance sampling method in PINNs for solving high dimensional partial differential equations*, J. Comput. Phys., 521 (2025), 113561, https://doi.org/10.1016/j.jcp.2024.113561.