Overlapping Schwarz preconditioners for randomized neural networks with domain decomposition

Shang, Yong; Heinlein, Alexander; Mishra, Siddhartha; Wang, Fei

DOI
10.1016/j.cma.2025.118011

Publication date
2025

Document Version
Final published version

Published in
Computer Methods in Applied Mechanics and Engineering

Computer Methods in Applied Mechanics and Engineering 442 (2025) 118011

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

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

Overlapping Schwarz preconditioners for randomized neural networks with domain decomposition

Yong Shang [a,1], Alexander Heinlein [b], Siddhartha Mishra [c,d,e], Fei Wang [a,*,2]

[a] School of Mathematics and Statistics, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, PR China
[b] Delft Institute of Applied Mathematics, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands
[c] Seminar for Applied Mathematics, ETH Zürich, Rämistrasse 101, 8092 Zürich, Switzerland
[d] D-MATH, ETH Zürich, Rämistrasse 101, 8092 Zürich, Switzerland
[e] ETH AI Center, ETH Zürich, Rämistrasse 101, 8092 Zürich, Switzerland

ARTICLE INFO

Keywords:
Randomized neural networks
Domain decomposition
Least-squares method
Principal component analysis
Overlapping Schwarz preconditioners

ABSTRACT

Randomized neural networks (RaNNs), characterized by fixed hidden layers after random initialization, offer a computationally efficient alternative to fully parameterized neural networks trained using stochastic gradient descent-type algorithms. In this paper, we integrate RaNNs with overlapping Schwarz domain decomposition in two primary ways: firstly, to formulate the least-squares problem with localized basis functions, and secondly, to construct effective overlapping Schwarz preconditioners for solving the resulting linear systems. Specifically, neural networks are randomly initialized in each subdomain following a uniform distribution, and these localized solutions are combined through a partition of unity, providing a global approximation to the solution of the partial differential equation. Boundary conditions are imposed via a constraining operator, eliminating the necessity for penalty methods. Furthermore, we apply principal component analysis (PCA) within each subdomain to reduce the number of basis functions, thereby significantly improving the conditioning of the resulting linear system. By constructing additive Schwarz (AS) and restricted AS preconditioners, we efficiently solve the least-squares problems using iterative solvers such as the Conjugate Gradient (CG) and generalized minimal residual methods. Numerical experiments clearly demonstrate that the proposed methodology substantially reduces computational time, particularly for multi-scale and time-dependent PDE problems. Additionally, we present a three-dimensional numerical example illustrating the superior efficiency of employing the CG method combined with an AS preconditioner over direct methods like QR decomposition for solving the associated least-squares system.

* Corresponding author.
E-mail addresses: fsy2503@stu.xjtu.edu.cn (Y. Shang), a.heinlein@tudelft.nl (A. Heinlein), smishra@sam.math.ethz.ch (S. Mishra), feiwang.xjtu@xjtu.edu.cn (F. Wang).

[1] The work of this author was partially supported by the National Natural Science Foundation of China (Grant No. 12375175).
[2] The work of this author was partially supported by the National Natural Science Foundation of China (Grant No. 92470115) and Tianyuan Fund for Mathematics of the National Natural Science Foundation of China (Grant No. 12426105).

https://doi.org/10.1016/j.cma.2025.118011
Received 26 December 2024; Received in revised form 6 April 2025; Accepted 7 April 2025
Available online 25 April 2025

1. Introduction

Domain decomposition methods (DDMs) are based on dividing the computational domain into overlapping or non-overlapping subdomains, breaking the problem into smaller subproblems. This allows local subproblems to be solved efficiently on multiple processors. DDMs greatly improve the convergence rates and computational efficiency of solving partial differential equations (PDEs), making them highly effective for a wide range of complex problems. Recently, the combination of DDMs with neural networks for solving partial differential equations has attracted significant attention, as it can offer significant computational speed-ups robustness and enhanced accuracy for various deep learning models. These hybrid approaches are often classified based on the machine learning algorithms they incorporate [1,2]. In this section, we aim to provide a brief overview of combining DDMs with two prominent machine learning models.

1.1. Physics-informed neural networks

In 1994, Dissanayake and Phan-Thien [3] introduced a neural network-based method that transforms solving PDEs into an unconstrained minimization problem, offering an efficient solution method. Later, Lagaris et al. [4] developed a neural network approach for initial and boundary value problems by designing a trial solution that automatically satisfies boundary conditions and optimizes the network to solve the differential equation.

Based on those pioneering works, Physics-informed neural networks (PINNs) [5] have recently been widely used to solve a variety of PDEs by using neural networks. Consider the following generic boundary value problem (BVP) on the domain $\Omega \subset \mathbb{R}^d$,

$$\begin{aligned} \mathcal{A}[u] &= f \qquad \text{in } \Omega, \\ \mathcal{B}[u] &= g \qquad \text{on } \partial\Omega, \end{aligned} \tag{1.1}$$

where $u : \Omega \to \mathbb{R}$ represents the solution, $\mathcal{A}$ is a differential operators, $\mathcal{B}$ is the boundary conditions operator, and $f : \Omega \to \mathbb{R}$ is a given right-hand side function. The regularity of u and f depends on $\mathcal{A}$ and $\mathcal{B}$.

To solve (1.1) using PINNs, we aim to find a neural network $u_p(x,\theta) : \Omega \to \mathbb{R}$ that approximates the solution u. In the case of a fully connected feedforward network, $u_p(x,\theta)$ is given by

$$u_p(x,\theta) = f_n \circ f_{n-1} \circ \ldots \circ f_i \circ \ldots \circ f_1(x,\theta),$$

where $f_i(x,\theta_i) = \sigma_i(A_i x + b_i)$ for $i = 1,\ldots,n-1$ and $f_n(x,\theta_n) = A_n x + b_n$. Here, $A_i \in \mathbb{R}^{d_i \times d_{i-1}}$, $b_i \in \mathbb{R}^{d_i}$, d_i is the number of neurons in the ith layer. σ_i denotes the activation function in the ith layer, which can be chosen as the commonly used tanh function for all layers. d_0 and d_n represent the input and output dimensions, respectively, which must align with the dimensions of the problem. Moreover, $\theta_i = (A_i, b_i)$ is the set of learnable parameters in the ith layer, and $\theta = (\theta_1,\ldots,\theta_n)$ are all learnable parameters.

By constructing loss functions based on the residual terms of the equations in (1.1), we employ a gradient descent method, such as the Adam optimizer, to minimize the loss function. The loss function is a discretized version of $\mathcal{L}(\theta)$,

$$\mathcal{L}(\theta) = \int_\Omega \big(\mathcal{A}[u_p](x,\theta) - f(x)\big)^2 dx + \lambda \int_{\partial\Omega} \big(\mathcal{B}[u_p](s,\theta) - g(s)\big)^2 ds, \tag{1.2}$$

where $\lambda > 0$ is a scalar weight used to balance the terms in the loss function.

Alternatively, the neural network can take the form of the solution ansatz [4,6–9]. Take Dirichlet boundary conditions as an example. One can adapt the constraining operator C such that

$$Cu_p(x,\theta) = L(x)u_p(x,\theta) + G(x) \tag{1.3}$$

satisfies the boundary conditions automatically. Here, $L(x)$ and $G(x)$ are some constructed functions chosen such that (1.3) meets the boundary conditions defined in (1.1). For other types of boundary conditions, similar constructions can be found in [7,8]. Then, $\mathcal{L}(\theta)$ becomes

$$\mathcal{L}(\theta) = \int_\Omega \big(\mathcal{A}[Cu_p](x,\theta) - f(x)\big)^2 dx, \tag{1.4}$$

which eliminates the need for penalty parameter λ in (1.2). However, now the operator C has to be chosen appropriately.

Extending the idea of PINNs, conservative PINNs [10] adopt a non-overlapping domain decomposition method specifically designed for conservation laws by enforcing the flux continuity in the strong form along the subdomain interfaces. This approach is further developed in [11] as extended PINNs (XPINNs) to handle general PDEs and arbitrary space–time domains. Furthermore, hp-variational PINNs (hp-VPINNs) [12] utilize piecewise polynomials as test functions within each subdomain, thereby improving the precision of solutions across the computational domain. Another approach, the deep domain decomposition method (Deep-DDM) [13], leverages a classical fixed-point Schwarz iteration [14] to decompose the boundary value problem. Finite basis PINNs (FBPINNs) were introduced in [15], where overlapping Schwarz domain decomposition is used to construct a global solution from localized NN-based functions. This concept was further explored in [16], which applied additive, multiplicative, and hybrid iteration techniques to train FBPINNs. To solve problems with high frequency and multi-scale solutions, multilevel FBPINNs were introduced in [17], incorporating multiple levels of domain decomposition. Additionally, [18] adopted a similar approach and enhanced the efficiency of time-dependent problems, and [19] developed a DDM for Kolmogorov–Arnold Networks (KANs), enabling the efficient training of KANs to solve multiscale problems.

1.2. Randomized neural networks

A major advantage of PINN-based methods is their mesh-free nature; they eliminate the need for discretizing PDEs, as the differential operators in the governing equations are approximated using automatic differentiation. Nevertheless, these methods still encounter challenges concerning accuracy and computational cost, primarily stemming from the inherently non-convex optimization problem [20], which often results in convergence to local minima [10].

For this reason, Pao et al. [21] introduced random vector functional-links (RVFLs), a method for single-hidden layer networks where connections to the hidden layer are fixed after random initialization, leaving only the output layer weights adjustable. This approach reduces training costs, improves computational efficiency [22], and maintains universal approximation capabilities [23]. Extreme Learning Machines (ELMs) [24] further advanced this concept by analytically computing output weights through solving a linear system, achieving fast learning speeds while maintaining universal approximation properties [25]. A detailed survey of randomized neural networks (RaNNs) was later presented in [26].

Taking the single-hidden layer as an example, a RaNN $u_r(x, W) : \Omega \to \mathbb{R}$ is given by:

$$u_r(x, W) = W \cdot \sigma(Rx + b), \tag{1.5}$$

where $R \in \mathbb{R}^{m\times d}$ and $b \in \mathbb{R}^m$ are randomly generated from a uniform distribution and fixed thereafter, and $W \in \mathbb{R}^m$ is the output weight that needs to be determined. The function σ is the activation function.

To solve (1.1) using (1.5), given a set of collocation points $\{x_i\}_{i=1}^N \subset \Omega$, and a constraining operator C defined in (1.3), the goal is to find W such that ideally

$$\mathcal{A}[L(x_i)W \cdot \big(\sigma(Rx_i + b)\big) + G(x_i)] = f(x_i) \quad \forall i \in \{1, \dots, N\}. \tag{1.6}$$

If $\mathcal{A}$ is a linear operator, then we define matrix $E \in \mathbb{R}^{N\times m}$ with $E_{ik} = \mathcal{A}[L(x_i)\sigma(R_k x_i + b_k)]$ and a vector $T \in \mathbb{R}^N$ with $T_i = f(x_i) - \mathcal{A}[G(x_i)]$, then the above Eq. (1.6) can be written as

$$EW = T. \tag{1.7}$$

Here, E is generally not invertible. Instead of using gradient-based algorithms like the Adam optimizer, the vector W can be determined by the least-squares solutions of the system (1.7). This approach provides a fast alternative to traditional neural network training techniques [24]. Based on ELMs, using an under-determined collocation scheme (where the number of neurons exceeds the number of points), was proposed for approximating solutions to elliptic PDEs with sharp gradients in [27].

1.3. Domain decomposition for randomized neural networks

Domain decomposition methods are numerical strategies that decompose a global computational domain into smaller subdomains, which may be either overlapping or non-overlapping. Although much of the computational workload within each subdomain can be performed independently, a certain degree of communication between subdomains remains essential to achieve a globally consistent solution; comprehensive introductions can be found in, for instance, [28–30].

In recent years, several innovative approaches combining non-overlapping domain decomposition methods with ELMs/RaNNs have emerged. In [31], local ELMs are employed to represent solutions within each subdomain, enforcing continuity constraints via the penalty method at collocation points along subdomain boundaries. Nevertheless, studies in [32,33] revealed a critical limitation of conventional ELM approaches: achieving high accuracy necessitates a significantly large number of neurons in the final hidden layer. To address this limitation, [34] introduced a modified method termed hidden-layer concatenated ELM, inspired by residual neural network architectures. This strategy employs hidden-layer concatenation, improving solution accuracy for both linear and nonlinear partial differential equations.

In a related line of research, Schiassi et al. [9] combined the theory of functional connections with ELMs to construct solutions for PDEs that analytically satisfy prescribed boundary conditions. This framework enabled the resolution of a diverse range of PDE problems with remarkable accuracy and reduced computational cost. Later, Wang and Dong [35] adopted this strategy and leveraged ELM-based RaNNs to solve high-dimensional PDEs. Moreover, Sun et al. [36,37] proposed combining RaNNs with discontinuous Galerkin methods, while Dang and Wang [38] explored a hybrid discontinuous Petrov–Galerkin framework integrated with RaNNs. Additionally, overlapping DDMs employing random feature functions were presented in [39], utilizing a collocation method coupled with penalty techniques to accurately handle boundary conditions.

In terms of accuracy and computational efficiency, all proposed approaches demonstrate significant advantages over existing DNN-based PDE solvers. However, certain aspects still necessitate further refinement. Firstly, as mentioned in [34], a part of the degrees of freedom provided by the hidden-layer nodes are to some extent “wasted”. Moreover, [27] indicate that the least-squares approach can result in ill-conditioned linear systems. Similar observations have been noted in [39,40], which emphasize increased computational complexity arising from significantly elevated condition numbers when utilizing least-squares frameworks. Furthermore, Chen et al. [41] conducted comparative studies involving direct and iterative methods for solving optimization problems associated with ELM frameworks. They specifically explored two prevalent preconditioning techniques — diagonal preconditioning and incomplete Cholesky factorization — to enhance the effectiveness of iterative solutions. The study concluded that existing iterative methods exhibit limited accuracy, underscoring the necessity of further advancements in preconditioning strategies, an area currently underrepresented in the literature.

In this paper, we focus on addressing the two aforementioned open challenges in the combination of domain decomposition methods and ELMs/RaNNs. Most importantly, we aim at efficiently and robustly solving ELMs/RaNNs, potentially enabling scalability to large-scale problems. In such scenarios, direct solution methods become impractical, necessitating iterative solvers whose convergence strongly depends on the spectral properties of the system matrix.

We tackle this by the combination of an overlapping domain decomposition-based neural network architecture, which localizes the basis functions and therefore results in a system matrix with sparse structure; our architecture is closely related to that of [39]. At the same time, the complexity of each individual local RaNN can be chosen significantly lower than that of a typical global RaNN model. The resulting sparse structure allows for the construction of one-level overlapping Schwarz preconditioners to improve the conditioning of the system matrix. It is noteworthy that limited prior research exists on applying domain decomposition methods to least-squares problems involving classical discretizations; refer to [42,43] for a few examples.

We observe that the one-level Schwarz preconditioner bounds the eigenvalues from above. However, significant (near-)linear dependencies among the hidden-layer basis functions persist and become more pronounced due to the increased overlap in the domain decomposition-based network architecture. To further alleviate these dependencies and enhance system matrix conditioning, we implement dimensionality reduction via principal component analysis (PCA). This technique not only significantly improves the conditioning but also effectively reduces redundant degrees of freedom. Importantly, PCA achieves this without notably compromising the neural network's approximation capabilities.

1.4. Singular value decomposition

Building upon the original ELM algorithm, a pruned-ELM approach was introduced in [44] to achieve more compact networks while preserving their generalization capabilities. This method identifies the relevance of hidden nodes to class labels and eliminates those that contribute little or nothing. Later, [45] extended this approach to regression problems, ranking neurons using multi-response sparse regression and selecting the optimal number of neurons through leave-one-out validation. Incremental extreme learning machines were proposed in [46,47] to dynamically adjust the number of neurons, adding or removing hidden-layer neurons as needed. Additionally, [48] applied a genetic algorithm to prune redundant or similar hidden neurons while maintaining performance comparable to standard classifiers.

Furthermore, the use of the singular value decomposition (SVD) in neural networks has been explored to reduce the model size while keeping the accuracy in recent research. One approach is to examine the SVD of the weight matrices in multi-layer neural networks. In [49], the SVD of the weight matrix was analyzed to show how the dimension of the weight space changes during backpropagation training. A method for reducing redundant hidden units by analyzing the matrix rank of the three-layered feedforward neural network using SVD was studied in [50], enabling network simplification without increasing the training error. Then, an online algorithm for determining the smallest possible number of neurons based on an SVD was proposed in [51]. In [52], for multi-layer neural networks with a large number of parameters, an SVD is applied to the weight matrices

$$\mathrm{A}_{b\times c} = U_{b\times e}\Sigma_{e\times e}V^T_{e\times c},$$

decomposing them into two smaller matrices, U and ΣV^T thereby reducing the number of parameters from bc to $(b+c)e$ when e is generally much smaller than c.

On the other hand, once the hidden weights of a neural network are fixed [53], a least-squares problem can be formulated to calculate the weights for the output layer, and then the SVD is computed on the hidden layer output matrix [53,54]. Psichogios and Ungar [53] proposed the SVD-NET algorithm to identify and eliminate redundant hidden neurons in a single hidden layer network. In [54], the authors find that the number of small singular values changes for different initialization of the weights. To address this, they analyzed the singular values across various initial weight settings and selected the optimal range of the initial weights. Teoh et al. [55] quantified the significance of the number of hidden neurons using SVD and proposed several possible methods for selecting the threshold parameter to eliminate the small singular values, illustrating that retraining the pruned network by an appropriate threshold would be more efficient than training the original network, as the training process aims to maximize the linear independence of the reduced set of hidden layer neurons.

In this paper, we use RaNNs with domain decomposition to solve PDEs. Overlapping Schwarz DDMs are utilized both to construct the global solution using RaNNs and to precondition the least-squares problem. In detail, the computational domain is partitioned into several subdomains, with RaNNs generated for each subdomain and initialized using a uniform distribution. Local window functions and the constraining operator are employed to construct a global approximate solution without the need for additional penalty terms. Then, a linear system $HW = F$ is constructed by selecting a set of collocation points within the domain to enforce the PDEs at these points. To improve the conditioning of the matrix H, defined as $\mathrm{cond}(H) = \frac{\sigma_{max}}{\sigma_{min}}$, where σ_{max} and σ_{min} represent the maximum and minimum singular values of H, respectively, we aim to eliminate small singular values, thereby increasing σ_{min} to improve the conditioning of the system. The maximum singular value is bounded in terms of a coloring constant, see, e.g., [56]. Principal Component Analysis (PCA) [57] is a statistical method used for dimensionality reduction and data compression by identifying principal components, which are linear combinations of the original variables, and eliminating correlations between the data. Here, PCA is applied to the hidden layer output matrix of RaNNs using SVD, resulting in neural networks that require fewer parameters and significantly eliminate small singular values. Finally, we design overlapping Schwarz preconditioners to efficiently solve a least-squares problem using a preconditioned iterative method like conjugate gradient (CG) and generalized minimal residual (GMRES) method. Meanwhile, we observe that employing the preconditioned conjugate gradient (PCG) method with Schwarz preconditioning significantly reduces the condition number.

1.5. Main contributions and structure of the paper

The principal contributions of this paper center around combining a domain decomposition-based neural network architecture, which inherently promotes sparsity, with advanced preconditioning techniques to enable efficient iterative solutions for ELM/RaNN-based least squares problems. While the architecture itself is related to prior work, particularly [39], the novel aspects of our approach include:

1. **First application of domain decomposition Schwarz preconditioners to randomized neural networks:** To the best of our knowledge, this paper represents the first effort to apply overlapping Schwarz preconditioning specifically to RaNNs to mitigate issues related to ill-conditioning. Previous studies primarily utilized domain decomposition to shape neural network architectures or to facilitate classical Schwarz fixed-point iterations. The locality of the neural network basis functions, which yields a sparse system matrix and enables the use of domain decomposition preconditioners, is enabled by the underlying overlapping domain decomposition-based neural network architecture.
2. **Integration of PCA with RaNNs to address near-linear dependencies among basis functions:** We propose applying PCA to the randomized basis functions within subdomains, effectively capturing the essential features of the neural network. This approach substantially reduces the number of trainable parameters and eliminates small singular values, thereby significantly enhancing matrix conditioning without compromising the neural network's core approximation capabilities.

The paper is organized as follows: Section 2 introduces the fundamental concepts of RaNNs integrated with overlapping Schwarz DDMs. Section 3 provides a detailed explanation of additive and restricted additive Schwarz preconditioners. Section 4 illustrates how the PCA process is formulated for RaNNs. Section 5 presents numerical results to demonstrate the accuracy and efficiency of the proposed framework. The final section concludes the paper with a discussion of the findings and potential future work.

2. RaNNs with domain decomposition

In overlapping Schwarz domain decomposition methods, the computational domain Ω is divided into J overlapping subdomains $\{\Omega_j\}_{j=1}^J$ classically. The global discretization space is often defined first, and then the local spaces are constructed as subspaces according to the domain decomposition [28,29]. Here, we define local spaces V_j of neural network functions and construct the global discretization space V from these local spaces using window functions.

Let $u_j(x, W_j)$ be a RaNN with a fully connected feedforward structure defined on subdomain Ω_j given by

$$u_j(x, W_j) = W_j \cdot f_j^{n-1} \circ \ldots \circ f_j^i \circ \ldots \circ f_j^1(x) := W_j \cdot \boldsymbol{\Phi}_j(x), \tag{2.8}$$

where $\boldsymbol{\Phi}_j(x) = f_j^{n-1} \circ \ldots \circ f_j^i \circ \ldots \circ f_j^1(x)$ is fixed and contains no learnable parameters. Each layer is defined as $f_j^i(x) = \sigma_i(R_i x + b_i)$, for $i = 1, \ldots, n-1$, and R_i, b_i are randomly generated from a given uniform distribution and remain fixed. The activation function σ_i is chosen as tanh for all layers. Therefore, only $W_j \in \mathbb{R}^m$ needs to be determined.

Then, we can define a function space of RaNNs as follows:

$$V_j = \text{span}\{\phi_j^1(x), \ldots, \phi_j^m(x)\}, \tag{2.9}$$

where $\phi_j^k(x)$ denotes the output of the kth neuron in the $n-1$ layer, and they are the components of $\boldsymbol{\Phi}_j(x)$, that is, $\boldsymbol{\Phi}_j = (\phi_j^1, \ldots, \phi_j^m)$. $\{\phi_j^k\}_{k=1}^m$ can be regarded as a set of basis functions. These functions generally have global support, hence, we need to localize them using window functions.

Therefore, we introduce $\{\omega_j\}_{j=1}^J$ as smooth window functions that form a partition of unity, confining the neural networks to their respective subdomains. In particular, we choose ω_j such that

$$\text{supp}(\omega_j) \subset \overline{\Omega}_j \quad \text{and} \quad \sum_{j=1}^J \omega_j \equiv 1 \quad \text{on } \Omega. \tag{2.10}$$

Similar to multilevel FBPINNs paper [17], we can define the global space of RaNN functions as

$$V = \sum_{j=1}^J \tilde{V}_j. \tag{2.11}$$

Here, the localized neural network function space is defined as

$$\tilde{V}_j = \text{span}\{\psi_j^1(x), \ldots, \psi_j^m(x)\}, \tag{2.12}$$

where

$$\psi_j^k(x) = \omega_j \phi_j^k(x), \quad j = 1, \ldots, J, \ k = 1, \ldots, m. \tag{2.13}$$

can be regarded as local basis functions. In other words, $\tilde{V}_j = \omega_j V_j$. An example of window functions ω_j and local basis functions ψ_j^k is shown in Fig. 1.

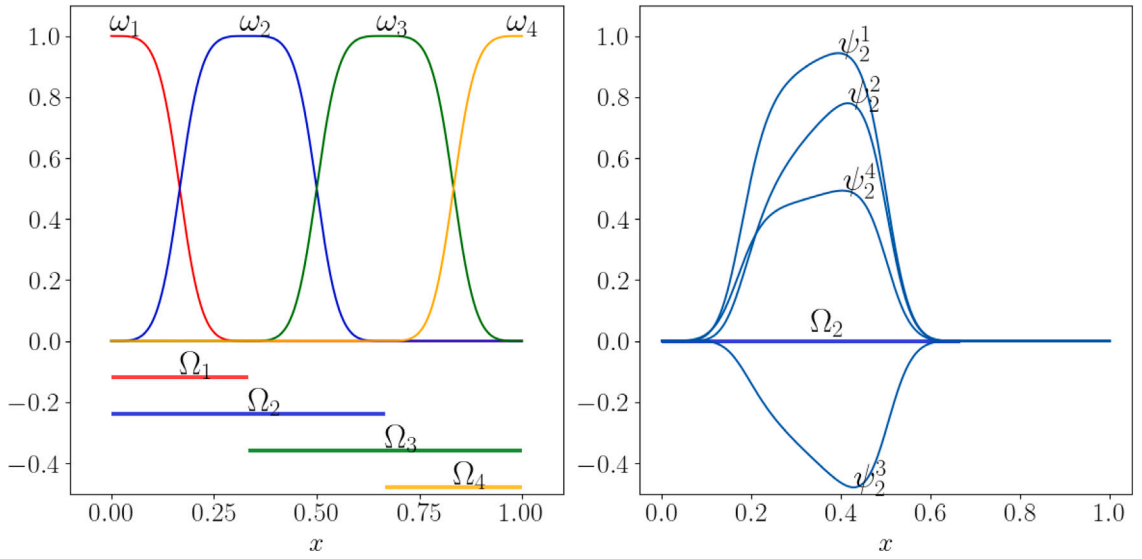


Fig. 1. Example of using RaNNs with overlapping Schwarz DDM to solve PDEs in one dimension. The domain is divided into four subdomains, and RaNNs with one hidden layer and four hidden neurons are used for each subdomain. The window functions w_j, each with local support within Ω_i, are displayed in the left figure. Exemplary local basis functions $\psi_j^k(x)$ for the second subdomain are shown in the right figure, where the global basis functions are multiplied by the window functions ω_2 to obtain the local basis functions $\psi_2^k(x)$.

Thus, the approximated solution $\hat{u}(x)$ in V can be represented as a combination of contributions from networks on all subdomains, multiplied by a constraining operator C defined in (1.3) to enforce the boundary conditions,

$$\hat{u}(x) = C\sum_{j=1}^{J}\omega_j u_j(x) = L(x)\sum_{j=1}^{J}\omega_j W_j\cdot\boldsymbol{\Phi}_j(x) + G(x) = \sum_{j=1}^{J} W_j\cdot\big(L(x)\omega_j\boldsymbol{\Phi}_j(x)\big) + G(x). \tag{2.14}$$

where $L(x)$ and $G(x)$ are constructed functions related to the domain Ω, designed such that Eq. (2.14) satisfies the boundary conditions defined in (1.1).

By substituting Eq. (2.14) into (1.1), and selecting a set of uniformly-spaced collocation points $\{x_i\}_{i=1}^N \in \Omega$, we aim for the approximated solution $\hat{u}(x)$ to satisfy the PDEs within the global domain. We need to find W_j, $j = 1,\dots,J$, such that

$$\mathcal{A}\Big[\sum_{j=1}^{J} W_j\cdot\big(L(x_i)\omega_j\boldsymbol{\Phi}_j(x_i)\big)\Big] = f(x_i) - \mathcal{A}[G(x_i)],\quad \forall i\in\{1,\dots,N\}. \tag{2.15}$$

When $\mathcal{A}$ is a linear operator, we have

$$\sum_{j=1}^{J} W_j\cdot\mathcal{A}\big[L(x_i)\omega_j\boldsymbol{\Phi}_j(x_i)\big] = f(x_i) - \mathcal{A}[G(x_i)],\quad \forall i\in\{1,\dots,N\}. \tag{2.16}$$

The above equation yields

$$[H_1, H_2, \dots, H_J](W_1, W_2, \dots, W_J)^T = F, \tag{2.17}$$

where $H_j\in\mathbb{R}^{N\times m}$ with $(H_j)_{i,k} = \mathcal{A}[L\omega_j\phi_j^k(x_i)]$, and $F\in\mathbb{R}^N$ with $f(x_i) - \mathcal{A}[G(x_i)]$.

Remark 2.1. In the case where $\mathcal{A}$ is a nonlinear operator, nonlinear iterative methods such as Picard iteration and Newton's method can be employed, as discussed in [40]. In each iteration step, a linear least-squares problem is solved.

For simplicity, we rewrite the linear system (2.17) as

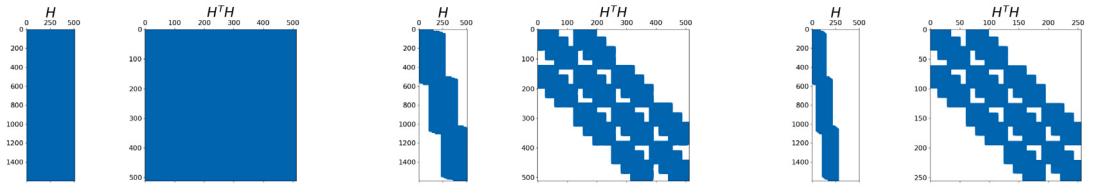
$$HW = F, \tag{2.18}$$

where $H = [H_1, H_2, \dots, H_J]$ and $W = (W_1^T, W_2^T, \dots, W_J^T)^T$. In general, the matrix H is not square (and not symmetric), and we solve it using the least-squares method. To solve the least-squares problem

$$\min \|HW - F\|_2^2,$$

the first-order optimality condition leads to the normal equation:

$$H^T HW = H^T F. \tag{2.19}$$



(a) Without DD, $\text{cond}(H^TH) = 10^{19}$. (b) DD, $m = 32$, $\text{cond}(H^TH) = 10^{16}$. (c) DD, $m = 16$, $\text{cond}(H^TH) = 10^{11}$.

Fig. 2. The sparsity patterns of the matrix and its condition number for different cases are shown: (a) without DD; (b) and (c) with a 4×4 DD. Different values of m are tested in (b) and (c), where (a) and (b) have the same number of total number of parameters, and (c) uses half of them.

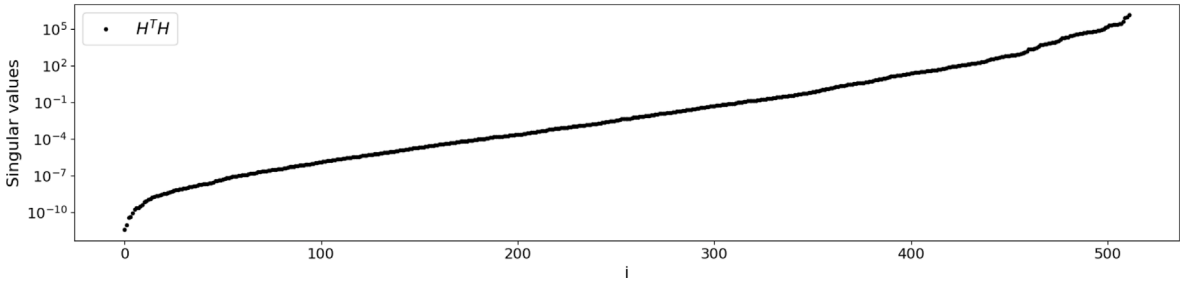


Fig. 3. Distribution of all 512 singular values of the matrix H^TH, arranged in non-decreasing order.

Here, $H^TH \in \mathbb{R}^{p\times p}$ is a sparse symmetric matrix, and $p = Jm$. The condition number of H^TH is given by $\text{cond}(H^TH) = \text{cond}(H)^2$.

A large condition number arises from the linear dependencies among the local basis functions ψ_j^k, which can result in a rank-deficient matrix. This, in turn, leads to small singular values and an increased squared condition number for the normal equations, if used. We propose additive and restricted additive Schwarz preconditioners, and use conjugate gradient (CG) and generalized minimal residual (GMRES) methods to solve the problem, respectively.

3. Overlapping Schwarz preconditioners

Consider solving the two-dimensional Laplace Eq. (5.29) using a RaNN without DDMs. We use a single RaNN with 512 hidden neurons and 40×40 collocation points. The resulting hidden-layer output matrix is dense and rank-deficient, with a condition number approximately equal to 10^{19}, as illustrated in Fig. 2(a). This significant ill-conditioning arises due to linear dependencies among the randomly generated basis, substantially increasing the computational cost of solving large-scale problems.

Next, we apply RaNNs combined with overlapping Schwarz domain decomposition under the same setting. We divide the computational domain into 16 subdomains, employing RaNNs with 32 hidden neurons per subdomain, thus maintaining the total parameter count. Domain decomposition introduces locality into the basis functions, which are multiplied by window functions. As shown in Fig. 2(b), the resulting matrix becomes block-sparse, facilitating the construction of domain decomposition preconditioners analogous to classical discretizations.

Fig. 3 illustrates the singular value distribution of the matrix H^TH before applying preconditioning, revealing a condition number of 10^{16}. The smallest singular value is approximately 10^{-11}, and the largest around 10^6, leading to slow convergence when using iterative solvers such as CG and GMRES [58]. Consequently, designing an effective preconditioner becomes essential to improve convergence and mitigate system ill-conditioning.

Furthermore, reducing the number of hidden nodes to $m = 16$ (Fig. 2(c)) significantly improves the condition number but may compromise accuracy due to fewer basis. To address this, we propose a PCA-based strategy in Section 4 that efficiently removes redundant neurons while preserving the essential features of the neural network.

As Chen et al. mentioned in [41], solvers for least-squares problems based on RaNNs (or ELMs) have not been well-developed, particularly for ill-conditioned problems with sparse structures. The condition number of the matrix in RaNN-based methods is extremely large [40]. Therefore, we propose using preconditioners to accelerate convergence in iterative methods such as the CG and GMRES method. Schwarz preconditioners are widely employed in DDMs to enhance the efficiency of iterative solvers for PDEs; (see [28,59] for instance.) Therefore, we focus on constructing overlapping Schwarz preconditioners specifically for RaNN-based methods.

Based on (2.11) and (2.12),we can define restriction operators

$$R_i : V \to \tilde{V}_i, \quad R_i(\sum_{j=1}^{J} \omega_j u_j) = \omega_i u_i, \quad i = 1, \dots, J,$$

which take a function, expressed as a sum of functions from $\tilde{V}_i$, and maps it onto the term in $\tilde{V}_i$. Similarly, extension operators are

$$R_i^T : \tilde{V}_i \to V, \quad R_i^T(\omega_i u_i) = \omega_i u_i, \quad i = 1, \dots, J.$$

Then, an additive Schwarz (AS) preconditioner for $H^T H$ is given by

$$M_{AS}^{-1} = \sum_{i=1}^{J} R_i^T A_i^{-1} R_i, \tag{3.20}$$

where $A_i = R_i H^T H R_i^T$, for $i = 1, \dots, J$. In practice, let R_i be a $q \times p$ Boolean matrix, and let s_j denote the indices of the local basis functions $\psi_j^k(x)$ that lie within the interior of Ω_j. We can construct the matrix such that $R_i[:, s_j] = I_q$, where $I_q \in \mathbb{R}^{q\times q}$ is the identity matrix, and R_i^T is constructed analogously.

By reducing the communication between different subdomains, a more effective preconditioner called the restricted additive Schwarz (RAS) preconditioner was proposed in [60]. After introducing the algebraic partition of unity matrix decribed, e.g., in [30], denoted by $D_i \in \mathbb{R}^{n_i \times n_i}$, for $i = 1, \dots, J$, the RAS preconditioner can be represented as:

$$M_{RAS}^{-1} = \sum_{i=1}^{N} R_i^T D_i A_i^{-1} R_i, \tag{3.21}$$

Here, D_i is a diagonal nonnegative matrix that satisfies the relation

$$\sum_{i=1}^{J} R_i^T D_i R_i = I_p.$$

where $I_p \in \mathbb{R}^{p\times p}$ is the identity matrix.

The inverse A_i^{-1} is computed using the QR decomposition, which is numerically stable,

$$A_i = Q_i P_i, \quad i = 1, \dots, J,$$

where Q_i is an orthogonal matrix with $Q_i^T = Q_i^{-1}$ and P_i is an upper triangular matrix. Here, we call it P_i as we already use the symbol R_i. Then, we have

$$A_i^{-1} = P_i^{-1} Q_i^T, \quad i = 1, \dots, J,$$

where P_i^{-1} denotes the inverse of P_i when it is invertible; otherwise, the pseudo-inverse P^+ is used, with both computed using Gaussian elimination.

With Schwarz preconditioners, we aim to efficiently solve the preconditioned system:

$$M^{-1} H^T H W = M^{-1} F, \tag{3.22}$$

where the preconditioner M^{-1} can be chosen as either the additive Schwarz preconditioner M_{AS}^{-1} or the restricted additive Schwarz preconditioner M_{RAS}^{-1}. This preconditioned system significantly alleviates the ill-conditioning of the original problem and improves convergence speed when employing iterative methods such as the CG or GMRES algorithms.

Since both $H^T H$ and M_{AS}^{-1} are symmetric, when they are also positive definite, it is advantageous to use the preconditioned conjugate gradient (PCG) method instead of directly applying the CG method. By introducing $M_{AS}^{-1/2}$, we reformulate the system as:

$$M_{AS}^{-1/2} H^T H M_{AS}^{-1/2} Y = M_{AS}^{-1/2} F, \tag{3.23}$$

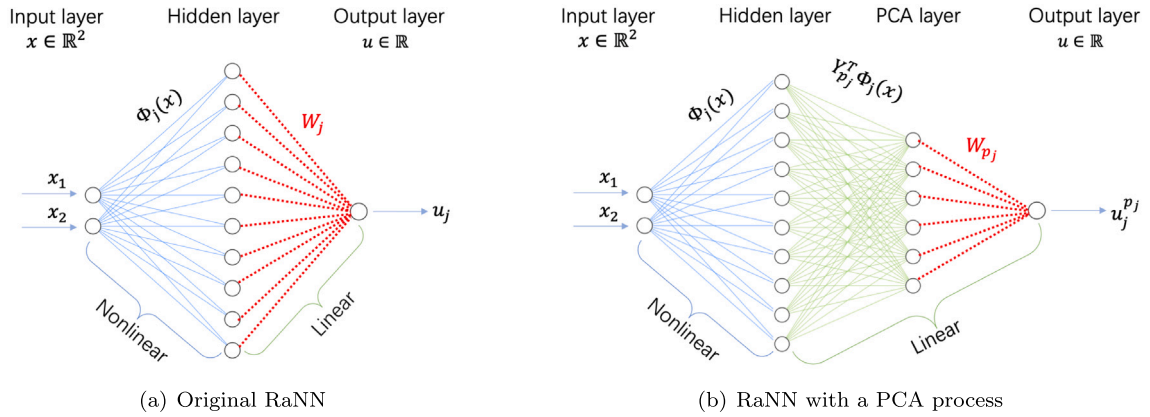
where $Y = M_{AS}^{-1/2} W$. In this case, $M_{AS}^{-1/2} H^T H M_{AS}^{-1/2}$ remains symmetric and positive definite, and its condition number closely relates to the eigenvalue distribution. Here, $M_{AS}^{-1/2}$ can be constructed using an incomplete Cholesky factorization, and a PCG algorithm can be implemented accordingly [61].

Remark 3.1. To obtain a matrix H with a relatively low condition number, we can carefully select parameters to minimize linear dependencies among the basis functions $\psi_j^k(x)$. Such linear dependencies typically result in very small singular values, thereby significantly increasing the condition number. One effective approach is to decrease the similarity of data representations of $\psi_j^k(x)$ within overlapping regions of subdomains. This can be practically accomplished by ensuring the number of data points N exceeds the number of hidden neurons m.

Moreover, integrating RaNNs with overlapping Schwarz DDMs naturally yields a block-sparse matrix structure, as depicted in Fig. 2. This sparse and structured form facilitates the straightforward construction of overlapping Schwarz preconditioners. In contrast, without employing DDMs, the resulting matrix would be dense, complicating the design of effective preconditioners.

4. PCA process

The phenomenon known as condensation of neural networks is discussed in [62,63], where it is observed that for networks with random initialization, the weights from an input node to hidden neurons quickly converge to similar values after training. This results in multiple hidden neurons can be replaced by an effective neuron with low complexity [63].



(a) Original RaNN (b) RaNN with a PCA process

Fig. 4. Network structure of the original RaNN $u_j : \mathbb{R}^2 \to \mathbb{R}$ and new output $u_j^{p_j}$ after a PCA process, the solid blue line represents the parameters of the neural network that are randomly initialized and fixed thereafter, and the solid green line represents the PCA process, which involves a matrix multiplication with $Y_{p_j}^T$. The dotted red line indicates the parameters that need to be determined. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

For RaNNs, we apply a PCA process as shown in Fig. 4(b), which preserves the principal components of the neural network, and reduces the number of solvable parameters while maintaining much of the network's approximation capability. This approach helps to mitigate linear dependencies within the system, resulting in a lower condition number.

From the construction of H, we know

$$H = [H_1, H_2, \dots, H_J], \quad (H_j)_{i,k} = \mathcal{A}[L\omega_j \phi_j^k(x_i)]. \tag{4.24}$$

Here, for a given problem (1.1) with operator $\mathcal{A}$, we can either perform an SVD on H_j directly or analyze the output matrix of the local basis functions $\psi_j^k(x)$ denoted by

$$(\Psi_j)_{i,k} = \psi_j^k(x_i), \ j = 1, \dots, J. \tag{4.25}$$

The latter approach enables us to focus on the key components in the network output that contribute to the solution approximation. It is problem independent and can be done in an offline process, ensuring a more efficient solution for different problem settings after initialization of the weights.

After performing a reduced singular value decomposition (SVD) on Ψ_j, we obtain

$$\Psi_j = U_j \Sigma_j Y_j^T, \quad j = 1, \dots, J,$$

where $k = \min(N_1, m)$, $U_j \in \mathbb{R}^{N_1 \times k}$, $\Sigma_j \in \mathbb{R}^{k \times k}$, and $Y_j \in \mathbb{R}^{m \times k}$.

Then, we simply select a constant threshold parameter τ to identify the singular values greater than τ, denoting their count as p_j on subdomain Ω_j. Next, we choose the first p_j-th columns of Y_j to form the truncated matrix $Y_{p_j} \in \mathbb{R}^{m \times p_j}$, respectively.

PCA process can be performed based on the SVD of Ψ_j, and the truncated hidden layer output matrix $T_j \in \mathbb{R}^{N_1 \times p_j}$ can be obtained by:

$$T_j = \Psi_j Y_{p_j}, \quad j = 1, \dots, J. \tag{4.26}$$

Now, we integrate a PCA process into the original RaNN, and the new output of the RaNN is given by:

$$\begin{aligned} u_j^{p_j}(x) &= \sum_{k=1}^{p_j} W_{p_j}^k \sum_{i=1}^{m} \Phi_j^i(x)(Y_{p_j})_{i,k} \\ &= \sum_{k=1}^{p_j} W_{p_j}^k \sum_{i=1}^{m} (Y_{p_j}^T)_{k,i} \Phi_j^i(x) \\ &= W_{p_j} \cdot [Y_{p_j}^T \Phi_j(x)]. \end{aligned} \tag{4.27}$$

Here, p_j denotes the number of effective neurons, and $W_{p_j} \in \mathbb{R}^{p_j}$ are the new weights that need to be determined, and $W_{p_j}^k$ denote the k-th row of W_{p_j}. The total number of parameters to be determined is given by $\sum_{j=1}^{J} p_j$, which does not exceed $p = mJ$.

In Fig. 4(a), the network structure of the original RaNN is illustrated, and in Fig. 4(b), the PCA process is shown, where $\Phi_j(x)$ represents the output of the nonlinear part. The term $Y_{p_j}^T \Phi_j(x)$ is referred to as the PCA layer, which is linear and only involves matrix multiplication, and does not introduce any additional unknown parameters.

Similar to (2.16), we obtain the following linear system, aiming to find W_{p_j}, $j = 1, \dots, J$, such that

$$\sum_{j=1}^{J} W_{p_j} \cdot \mathcal{A}\big(L\omega_j[Y_{p_j}^T \Phi_j(x)]\big) = f(x_i) - \mathcal{A}(G(x_i)), \quad \forall i \in \{1, \dots, N\}. \tag{4.28}$$

Finally, we can construct the AS and RAS preconditioners and use the CG or GMRES method to solve (4.28), respectively. The whole algorithm is summarized in Algorithm 1.

Algorithm 1 Overlapping Schwarz preconditioners for RaNNs with domain decomposition.

Step 1. Use overlapping Schwarz DDM to divide the domain Ω into J overlapping subdomains $\{\Omega_j\}_{j=1}^{J}$.
Step 2. Initialize J RaNNs with network architecture $u_j : \Omega_j \to \mathbb{R}$ using a uniform distribution.
Step 3. Construct a global approximate solution by applying window functions and a constraint operator.
Step 4. Use collocation points $\{x_i\}_{i=1}^{N}$ and threshold parameter τ to perform PCA process.
Step 5. Use collocation points $\{x_i\}_{i=1}^{N}$ to assemble the linear system (4.28).
Step 6: Construct the AS and RAS preconditioners and solve the problem (4.28) with iterative method.
Step 7. Solve the network parameters W_{p_j} to $u_j^{p_j}$, $j = 1, \dots, J$.

Remark 4.1. A weight initialization strategy based on frequency information for constructing the initial RaNN to solve PDEs with varying solutions is presented in [64]. Neuron growth and layer growth strategies were also proposed to enhance the neural network's performance. Specifically, layer growth introduces additional layers to better capture complex patterns when a single layer is insufficient. Additionally, [65] proposes a transferable neural network model based on shallow networks, making it easier to adapt the neural feature space across different PDEs in various domains. [66] analyzed the computational costs of ELM schemes with different strategies for increasing the number of neurons and input samples, and established their convergence rates. Their study demonstrated that spectral-type convergence can be achieved when both the number of input samples and neurons increase simultaneously.

In [67], three different activation functions and four different initialization strategies were tested, all of which demonstrated good accuracy. In this paper, the inputs to each subdomain network are normalized, and we use a uniform distribution for initialization. The activation function, tanh, is selected for all problems, as it is commonly used. An approximation result for using shallow RaNNs with tanh activation and uniformly distributed weights, specifically for Sobolev functions, is presented in [68]. Additionally, [27,69] investigate how to appropriately set the values for weights and biases to obtain an effective underlying approximating subspace. Further research into various weight initialization strategies and activation functions remains an important area of exploration.

Remark 4.2. In practice, the PCA process is influenced by various factors, including the initialization of weights in neural networks, the number of collocation points and the choice of threshold parameter τ. The impact of these factors is to reduce the condition number of the system, further investigation is needed to better understand their effects. This is out of the scope of the current work, and will be explored in future investigations.

5. Numerical results

5.1. Problems studied

In Example 5.1, we consider a two-dimensional multi-scale Laplacian problem. In Section 5.3.1, we evaluate the performance of overlapping Schwarz preconditioners without applying the PCA process and compare the effectiveness of different iterative solvers. Section 5.3.2 analyzes how various factors, such as window functions, constraint operators, strategies of collocation point selection, and activation functions, influence the preconditioning performance. In Section 5.3.3, we investigate the impact of applying the PCA process, while Section 5.3.4 compares our approach with multilevel FBPINNs to assess its overall effectiveness.

In Example 5.2, we solve a one-dimensional advection-diffusion equation in the space–time domain. Using the same settings and total number of parameters, we compare the results with *hp*-VPINNs and the penalty method in Section 5.3.5. Our approach demonstrates superior accuracy and efficiency, while also directly satisfying the boundary and initial conditions, thus eliminating the need for penalty parameters.

Example 5.3 presents a three-dimensional problem. In Section 5.3.6, we compare the results obtained using the direct least-squares solver via QR decomposition, the CG method without a preconditioner, and the PCG method with the AS preconditioner. The PCG method significantly reduces the condition number to an exceptionally low value while maintaining accuracy.

5.2. Common implementation details

All results are obtained using an NVIDIA GeForce RTX 3090 GPU. We use *scipy.linalg.lstsq* in Python, relying on QR decomposition to directly solve the least-squares problem, and adopt the CG method and its variants, as well as the GMRES method from *scipy.sparse.linalg*, to solve the normal equation corresponding to the least-squares problem. We average the results over ten experiments for each problem to account for the randomness in the methods.

Table 1
Relative L^2 error and number of iterations for different preconditioners by GMRES in Example 5.1.

DoF	N	$\kappa(H^TH)$	M^{-1}	$\kappa(M^{-1}H^TH)$	σ_{min}	σ_{max}	$Iter$	e_{L^2}
256	1600	10^{11}	$None$	10^{11}	10^{-6}	10^{5}	2560	8.94e−2
			M_{AS}^{-1}	10^{7}	10^{-3}	10^{4}	13	6.44e−3
			M_{SAS}^{-1}	10^{7}	10^{-4}	10^{3}	10	6.45e−3
			M_{RAS}^{-1}	10^{7}	10^{-4}	10^{3}	31	6.41e−3

5.3. Numerical examples

Example 5.1. We consider the multi-scale Laplacian problem from [17] with Dirichlet boundary condition,

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega = [0,1]^2, \\ u &= 0 \quad \text{on } \partial\Omega. \end{aligned} \tag{5.29}$$

and the exact solution given by

$$u(x_1, x_2) = \frac{1}{n}\sum_{i=1}^{n} \sin(\omega_i \pi x_1)\sin(\omega_i \pi x_2), \tag{5.30}$$

where $n = 1,2, \cdots 6$ denotes the problem complexity, and $\omega_i = 2^i, i = 1, 2, \ldots, n$. $n = 6$ means the solution contains six multi-scale components with exponentially increasing frequencies. The source term $f = \frac{2}{n}\sum_{i=1}^{n}(\omega_i\pi^2)\sin(\omega_i\pi x_1)\sin(\omega_i\pi x_2)$.

5.3.1. Overlapping Schwarz preconditioners

In this example, the domain Ω is divided into $J = l \times l$ overlapping subdomains, and a uniform rectangular domain decomposition is constructed. Each dimension is defined as follows:

$$\Omega_j = \left[\frac{(j-1)-\delta/2}{l-1}, \frac{(j-1)+\delta/2}{l-1}\right] \quad l \geq 1,$$

where $\delta > 1$ is defined as the ratio of the subdomain size and the overlap width between subdomains.

Same as in [17], the subdomain window functions are given by

$$\omega_j = \frac{\hat{\omega}_j}{\sum_{j=1}^{J^{(l)}} \hat{\omega}_j} \quad \text{where} \quad \hat{\omega}_j = \Pi_i^2[1 + \cos(\pi(x_i - \mu_i)/\sigma_i)]^2, \tag{5.31}$$

where $\mu_i = (i-1)/(l-1)$ and $\sigma_i = (\delta/2)/(l-1)$ represent the center and half-width of each subdomain along each dimension, respectively. The constraining operator C is chosen using

$$L(x) = \tanh(x_1/\rho)\tanh((1-x_1)/\rho)\tanh(x_2/\rho)\tanh((1-x_2)/\rho), \quad G(x) = 0 \tag{5.32}$$

with $\rho = (1/2)^n$, ensuring that $Cu(x) = L(x)u(x) + G(x)$ automatically satisfies the boundary condition.

First, we consider using a 4×4 domain decomposition with 40×40 collocation points across the domain Ω to solve the problem (5.29) for $n = 2$. The overlap ratio δ is fixed by 2. RaNNs with one hidden layer and 16 hidden neurons on each subdomain are initialized by a uniform distribution $\mathcal{U}(-1, 1)$ for weights and bias parameters. Inputs to each subdomain network are normalized to the range $[-1, 1]^2$ within the respective subdomains. The number of parameters that need to be solved is denoted as the degrees of freedom (DoF) and is given by DoF $= mJ$. We first illustrate the performance of the overlapping Schwarz preconditioners without applying the PCA process. We construct the AS preconditioner and denoted as M_{AS}^{-1}. For the RAS preconditioner, we consider two choices for the coefficients of D_i in M_{RAS}^{-1}: one is to use $(D_i)_{jj} = 1/\mathcal{M}_j$ denoted by M_{SAS}^{-1} (scaled) [30], where $\mathcal{M}_j$ is defined as the number of overlapping subdomains for each node j; the other is to use a Boolean matrix and denoted by M_{RAS}^{-1} (restricted).

In Table 1, $\kappa(H^TH)$ denotes the condition number of the matrix H^TH, and $\kappa(M^{-1}H^TH)$ denotes the condition number of the preconditioned matrix, where M^{-1} represents different types of preconditioners. We observe that the condition number of the matrix H^TH is approximately 10^{11}, and all three preconditioners significantly reduce the condition number around four orders of magnitude. σ_{min} and σ_{max} denote the minimum and maximum singular values, respectively. We can see that by eliminating small singular values, σ_{min} increases, improving the conditioning of the system.

As for the iterative solvers, M_{AS}^{-1} is symmetric while M_{SAS}^{-1}, M_{RAS}^{-1} are not. Therefore, the GMRES method is applicable to all of them, as it does not require symmetry. The iteration is stopped when the relative residual norm of the solution falls below a fixed tolerance, i.e.,

$$\|M^{-1}H^THW - M^{-1}H^TF\|_{L^2} / \|M^{-1}H^TF\|_{L^2} \leq 10^{-5}.$$

Compared to the case without preconditioning, all preconditioners enable GMRES to converge in significantly fewer iterations, while achieving similar accuracy due to the same tolerance in the stopping criteria.

Fig. 5 shows the eigenvalue distribution in the complex plane for different preconditioners. We observe that the distribution of the eigenvalues of the AS preconditioned matrix becomes more concentrated. Meanwhile, the largest eigenvalue of the SAS

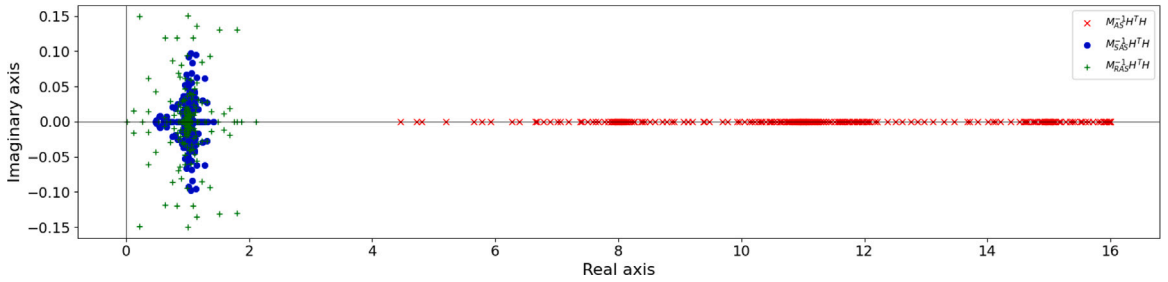


Fig. 5. The eigenvalue distribution of the preconditioned matrix with different preconditioners under a 4 × 4 domain decomposition in Example 5.1 is shown in the complex plane.

Table 2
Relative L^2 error and number of iterations for different preconditioners using various iterative solvers in Example 5.1.

Solver	$M^{-1} = I$		$M^{-1} = M_{AS}^{-1}$		$M^{-1} = M_{RAS}^{-1}$		$M^{-1} = M_{SAS}^{-1}$	
	Iter	e_{L^2}	*Iter*	e_{L^2}	*Iter*	e_{L^2}	*Iter*	e_{L^2}
CG	> 2000	1.95e−2	8	5.03e−3	–	–	–	–
CGS	> 2000	2.63e−2	4	5.04e−3	24	5.03e−3	6	5.04e−3
BICG	> 2000	1.03e−2	8	5.08e−3	32	5.05e−3	11	5.09e−3
GMRES	> 2000	8.68e−2	13	5.07e−3	31	5.06e−3	11	5.08e−3

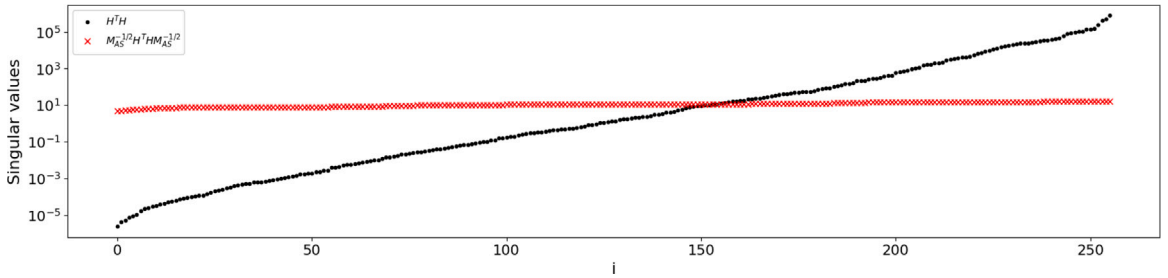


Fig. 6. The distribution of singular values for using the PCG method with AS preconditioner against the distribution without preconditioning ordered in non-decreasing order.

preconditioned matrix is much smaller than that of the AS preconditioned matrix, and its distribution becomes more concentrated than that of RAS. This is because the RAS preconditioned matrix includes a Boolean matrix, which is introduced to reduce communication between different subdomains. However, a potential risk for the RaNN-based method is that this may increase linear dependencies between local basis functions, as zero values occur in their common region, which may lead to a higher condition number than SAS. Consequently, the SAS preconditioner, which avoids this issue, provides better conditioning and requires smaller iteration steps to converge.

Besides, we try variants of the CG method to solve this problem under the same settings. In Table 2, both CG and its variants work with the AS preconditioner, allowing them to utilize fewer number of iteration steps compared to GMRES. Additionally, the Conjugate Gradient Squared (CGS) and Biconjugate Gradient (BICG) methods are effective with the SAS and RAS preconditioners when CG fails.

Specifically, $H^T H$ and M_{AS}^{-1} are symmetric and positive definite in this case, allowing us to apply the PCG method. The singular value distribution of $M_{AS}^{-1/2} H^T H M_{AS}^{-1/2}$ is shown in Fig. 6, where the condition number 10^{11} is reduced to 3.5. The minimum singular value increases from 10^{-7} to 4.6, while the maximum decreases from 10^5 to 16. In classical DDMs, the largest singular value is typically bounded by the maximum number of subdomains intersecting at a single collocation point—here, this value is 4, and due to the least-squares formulation, it becomes $4^2 = 16$. Consequently, the PCG method rapidly converges within just 8 iterations, achieving a relative L^2 error of approximately 5.02×10^{-3}.

5.3.2. Factors influence

Several factors may influence preconditioning performance, including strategies of collocation point selection, activation functions, window functions, and constraining operators. In this subsection, we provide detailed comparisons to illustrate their effects.

First, we assess how different collocation point selection strategies and activation functions impact the results under identical settings. Specifically, we compare three collocation point selection methods: random sampling from a uniform distribution, Gauss–Legendre quadrature points, and uniformly-spaced points. For activation functions, we consider tanh and sigmoid. We then analyze

Table 3
Relative L^2 error and number of iterations for different choices of collocation points and activation functions in Example 5.1.

Collocation points	Activation function	Without Preconditioner			AS precondtioner		
		Cond	*Iter*	e_{L^2}	*Cond*	*Iter*	e_{L^2}
Uniformly-spaced	tanh	10^{11}	3000	1.15e−2	3.5	8	5.02e−3
	sigmoid	10^{15}	3000	4.29e−2	3.6	8	3.38e−3
Quadrature points	tanh	10^{12}	3000	2.51e−2	28.3	21	9.03e−3
	sigmoid	10^{15}	3000	4.55e−2	49.6	22	5.86e−3
Random samples	tanh	10^{12}	3000	5.67e−2	38.3	22	3.18e−2
	sigmoid	10^{15}	3000	7.17e−2	46.7	25	1.68e−2

Table 4
Relative L^2 error and number of iterations for different window functions and constraining operators in Example 5.1.

Window function	Constraining operator	$M^{-1} = I$		$M^{-1} = M_{AS}^{-1}$	
$\hat{\omega}_j$	$L(x)$	*Iter*	e_{L^2}	*Iter*	e_{L^2}
$\Pi_i^2[1+\cos(\pi(x_i-\mu_i)/\sigma_i)]^2$	$x_1(1-x_1)x_2(1-x_2)$	5120	4.66e−2	18	3.43e−5
	$\sin(x_1)\sin(1-x_1)\sin(x_2)\sin(1-x_2)$	5120	5.30e−2	19	4.35e−5
	$(e^{x_1}-1)(e^{1-x_1}-1)(e^{x_2}-1)(e^{1-x_2}-1)$	5120	4.35e−2	20	4.62e−5
	$\tanh(x_1)\tanh(1-x_1)\tanh(x_2)\tanh(1-x_2)$	5120	4.20e−2	20	2.91e−5
$\Pi_i^2\, e^{-(\pi(x_i-\mu_i)/\sigma_i)^2}$	$x_1(1-x_1)x_2(1-x_2)$	5120	2.14e−1	8	1.88e−4
	$\sin(x_1)\sin(1-x_1)\sin(x_2)\sin(1-x_2)$	5120	2.68e−1	8	3.87e−4
	$(e^{x_1}-1)(e^{1-x_1}-1)(e^{x_2}-1)(e^{1-x_2}-1)$	5120	2.50e−1	7	2.75e−4
	$\tanh(x_1)\tanh(1-x_1)\tanh(x_2)\tanh(1-x_2)$	5120	2.33e−1	8	2.27e−4

the results obtained using the CG method without preconditioning and the PCG method with the AS preconditioner. The maximum number of iterations is set to 3000. The condition numbers for matrices associated with these methods are listed under *Cond*. As seen in Table 3, the combination of uniformly spaced collocation points and the tanh activation function yields a better-conditioned linear system. In all tested cases, the condition number significantly decreases to a remarkably low value with preconditioning.

Next, we examine the influence of different window functions and constraining operators on the preconditioning performance. We test four types of constraining operators, $Cu = Lu$, and two window functions, $\omega_j = \frac{\hat{\omega}_j}{\sum_j \hat{\omega}_j}$. We increase the number of hidden neurons to $m = 32$, and different $\hat{\omega}_j$ and $L(x)$ are provided in Table 4. Using the GMRES method without a preconditioner, we compare it with the AS preconditioners. The maximum number of iterations is set to 5120. In all cases, the AS preconditioner effectively reduces the condition number from approximately 10^{17} to 10^{12}, significantly decreasing the required iterations to achieve convergence. Furthermore, the choice of window functions and constraining operators does not significantly affect the results, offering an easy way to select them.

5.3.3. PCA results

To improve the accuracy, we can either increase the number of hidden neurons or use a larger overlap ratio. However, this also increases the condition number. For example, when setting $m = 32$ and $\delta = 2$, the matrix $H^T H$ has a large condition number around 10^{16}, and the relative L^2 error can be around 10^{-5}; when setting $m = 32$ and $\delta = 3$, the condition number increases further to approximately 10^{20}, and the relative L^2 error can be reduced to 10^{-7}. In this case, $H^T H$ would also be rank-deficient, and the preconditioned matrices would still exhibit a large condition number. By applying the PCA process, the condition number can then be significantly reduced, improving the convergence of the iterative methods.

In Table 5, let $m = 32$ and $\delta = 2$, we then examine how the threshold parameter τ for dropping small singular values based on the singular values influences the condition number and accuracy. It is varied from 10^{-4} to 10^{-1}. Within each subdomain Ω_j, we apply the same threshold parameter τ to determine the number of effective neurons p_j. Then, the number of parameters that need to be solved is given by DoF $= \sum_{j=1}^{J} p_j$. We use the GMRES method without a preconditioner, and compare with the AS and SAS preconditioners. We observe that, for both the AS and the SAS preconditioner, as the DoF decreases, the minimum singular value of preconditioned matrices grows. This leads to a lower condition number, which enables convergence within fewer iteration steps.

Additionally, we observe that setting $\tau = 10^{-3}$ helps to lower the condition number without significant accuracy loss. Fig. 7 illustrates the progression of the condition number and relative L^2 error with respect to τ, showing a substantial decrease in the condition number. The decrease is approximately two orders of magnitude, independent of τ. Meanwhile, the relative L^2 error increases gradually, and does not exceed the result obtained by solving the problem without a preconditioner.

5.3.4. Comparisons with multilevel FBPINNs

Finally, we consider using an increasing number of subdomains, $J = 2^n \times 2^n$, to solve the problem (5.29) with higher complexity, corresponding to the solution (5.30) for a larger value of n. Then compare against the results obtained from multilevel FBPINNs [17] with the weak scaling test shown in Fig. 8, which tested the same model problem. RaNNs with one hidden layer and 32 hidden

Table 5
Relative L^2 error and number of iterations for different preconditioners by GMRES in Example 5.1.

τ	DoF	$\kappa(H^TH)$	M^{-1}	$\kappa(M^{-1}H^TH)$	σ_{min}	σ_{max}	$Iter$	e_{L^2}
10^{-4}	512	10^{16}	$None$	10^{16}	10^{-10}	10^{6}	5120	3.72e−2
			M_{AS}^{-1}	10^{12}	10^{-6}	10^{6}	27	5.46e−5
			M_{SAS}^{-1}	10^{12}	10^{-7}	10^{5}	30	5.49e−5
$\mathbf{10^{-3}}$	**436**	$\mathbf{10^{13}}$	$None$	10^{13}	10^{-8}	10^{5}	4360	3.75e−2
			$\boldsymbol{M_{AS}^{-1}}$	$\mathbf{10^{10}}$	$\mathbf{10^{-5}}$	$\mathbf{10^{5}}$	**16**	**1.28e−4**
			M_{SAS}^{-1}	10^{10}	10^{-6}	10^{4}	18	1.28e−4
10^{-2}	335	10^{10}	$None$	10^{10}	10^{-5}	10^{5}	3350	4.51e−2
			M_{AS}^{-1}	10^{7}	10^{-3}	10^{4}	14	7.14e−4
			M_{SAS}^{-1}	10^{7}	10^{-4}	10^{3}	13	7.11e−4
10^{-1}	212	10^{8}	$None$	10^{8}	10^{-3}	10^{6}	2120	5.01e−2
			M_{AS}^{-1}	10^{4}	10^{-2}	10^{3}	12	7.13e−3
			M_{SAS}^{-1}	10^{4}	10^{-3}	10^{2}	11	7.10e−3

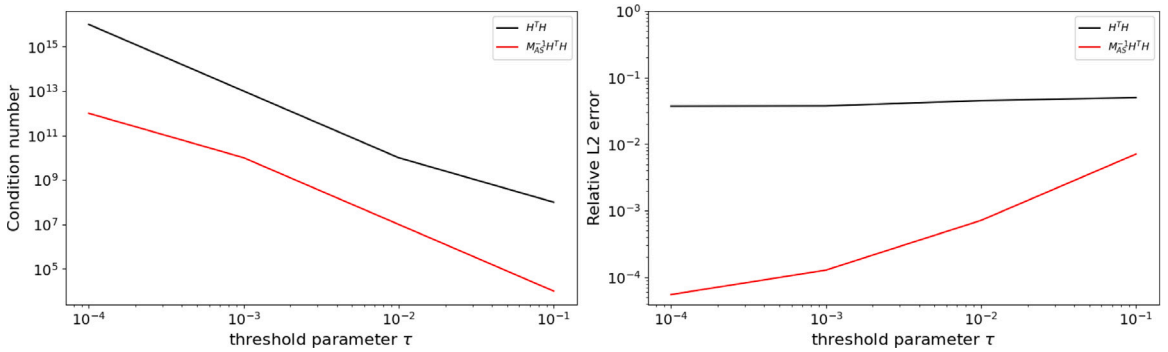


Fig. 7. The progressive of condition number and relative L^2 error via the threshold parameter τ by different preconditioners in Example 5.1.

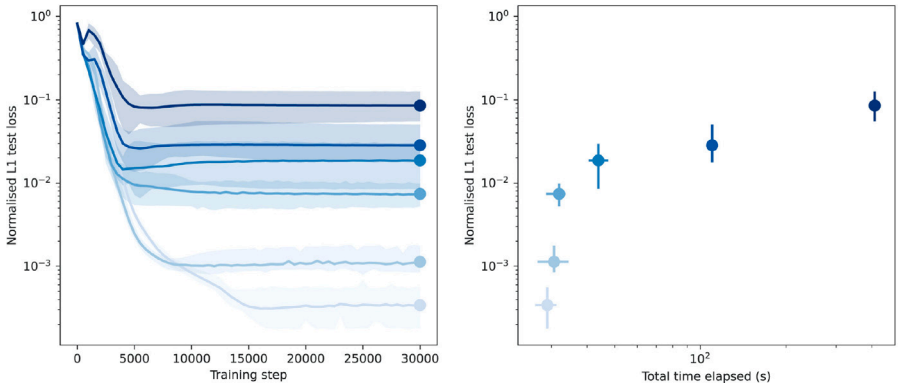


Fig. 8. Weak scaling test using multilevel FBPINNs (taken from [17]) in Example 5.1. The color-coded convergence curves and training times for each model are shown, and the color transitions from light to dark represent increasing problem complexity, ranging from $n = 1$ to $n = 6$.

neurons on each subdomain are employed. The number of collocation points is fixed at 100 in each subdomain, and the overlap ratio is fixed by $\delta = 2$. The threshold parameter τ is fixed by 10^{-3}. The normalized L^1 test loss e_{L^1} is computed approximately on $M = 350 \times 350$ uniformly-spaced test points by $\frac{1}{M}\sum_{i=1}^{M} \|\hat{u}(x_i) - u(x_i)\|_{l^1}/\gamma$, where γ is defined as the standard deviation of the true solutions set $\{u(x_i)\}_{i=1}^{M}$. We use the normalized L^1 test loss here to compare with the results in [17].

Table 6 shows the normalized L^1 test loss and the time for solving the preconditioned system using the GMRES method. We observe that both the AS and SAS preconditioners significantly reduce the condition number and result in fewer iterations, with the solution time not increasing drastically as the complexity n increases. However, the normalized L^1 test loss does reduce slightly as the problem complexity increases. The exact and numerical solutions for $n = 2, 3, \ldots, 6$ are shown in Fig. 9. Compared to the weak scaling test shown in Fig. 8 for the multilevel FBPINNs method, our approach achieves better accuracy with a smaller domain decomposition, Specifically, only 32×32 subdomains are required to handle a problem with complexity $n = 6$. Meanwhile, the

Table 6
Weak scaling test using GMRES method for different preconditioners in Example 5.1.

n	J	DoF	M^{-1}	$\kappa(M^{-1}H^TH)$	$Iter$	e_{L^1}	Time(s)
2	2 × 2	109	$None$	10^{13}	1090	6.81e−2	0.064
			M_{AS}^{-1}	1	1	1.98e−3	0.00040
			M_{SAS}^{-1}	1	1	1.97e−3	0.00033
3	4 × 4	436	$None$	10^{14}	4330	2.04e−1	0.81
			M_{AS}^{-1}	10^{9}	12	1.28e−2	0.00056
			M_{SAS}^{-1}	10^{9}	21	1.27e−2	0.00082
4	8 × 8	1706	$None$	10^{14}	17 060	1.92e−1	7.14
			M_{AS}^{-1}	10^{10}	38	2.46e−2	0.042
			M_{SAS}^{-1}	10^{10}	52	2.48e−2	0.056
5	16 × 16	6811	$None$	10^{14}	68 110	3.24e−1	116
			M_{AS}^{-1}	10^{10}	51	3.24e−2	0.31
			M_{SAS}^{-1}	10^{10}	60	3.22e−2	0.35
6	**32 × 32**	**27147**	$None$	10^{14}	271 470	4.36e−1	2950
			$\boldsymbol{M_{AS}^{-1}}$	$\mathbf{10^{11}}$	**109**	**6.71e−2**	**5.35**
			M_{SAS}^{-1}	10^{11}	138	6.50e−2	5.67

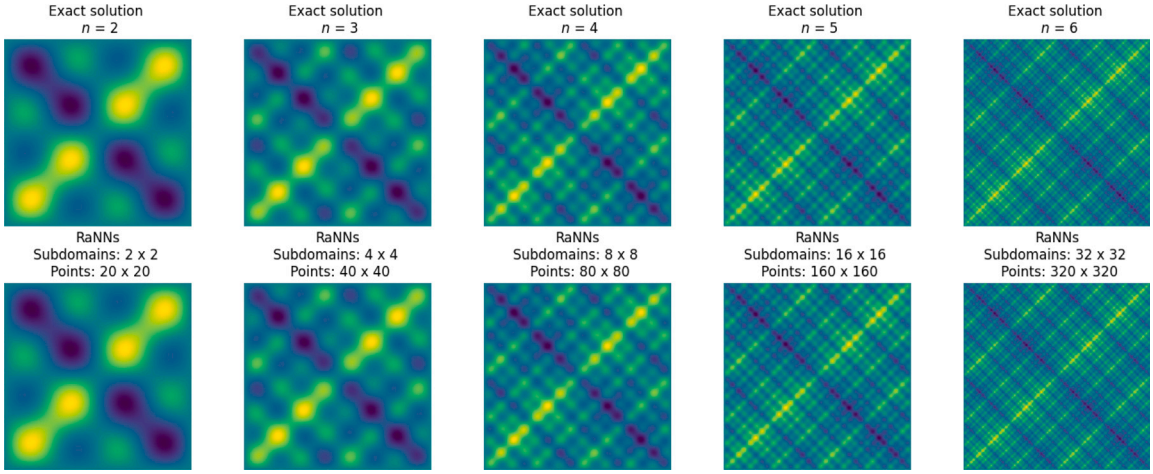


Fig. 9. Weak scaling test using GMRES method with AS preconditioner in Example 5.1. The top figure shows the exact solution for different problem complexity n, while the bottom figure presents the numerical results corresponding to each n.

time for solving the parameters of RaNNs using the GMRES method is much shorter than the training time for feedforward fully connected networks in multilevel FBPINNs.

Fig. 10 presents results from a weak scaling test using the AS preconditioner in Example 5.1. In the left plot, we observe the progression of the residual $\|H^THW - H^TF\|_{L^1}$ at each GMRES iteration for the problem (5.29) with $n = 2, 3, \ldots, 6$, illustrating the convergence behavior. The middle plot compares the GMRES result against the results obtained using QR decomposition to directly solve the least-squares problem. Our approach maintains comparable accuracy to solving the least-squares problem using QR decomposition while substantially reducing computational time. The right plot displays the results with a larger overlap ratio $\delta = 3$, where the normalized L^1 test loss is significantly reduced to approximately 10^{-3} without causing a major increase in computation time for each case. The total computational time includes the cost of constructing the linear system, building the preconditioner, and solving the preconditioned system using the GMRES method, with their respective percentages being approximately 60%, 38%, and 2% for $n = 6$. The primary part (linear system construction) will be further optimized in the future through parallelization.

Example 5.2. Given $\Omega = (-1, 1)$, we consider a one-dimensional advection-diffusion equation

$$\begin{aligned} \frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} &= \kappa \frac{\partial u^2}{\partial x^2} && \text{in } \Omega \times I, \\ u(-1, t) = u(1, t) &= 0 && \text{in } I, \\ u(x, 0) &= -\sin(\pi x) && \text{in } \Omega, \end{aligned}$$

with $I = (0, 1)$. Here, $\kappa = 0.1/\pi$ represents the diffusivity coefficient, which complicates the solution near the right boundary $x = 1$ as the no-slip boundary condition is imposed [12].

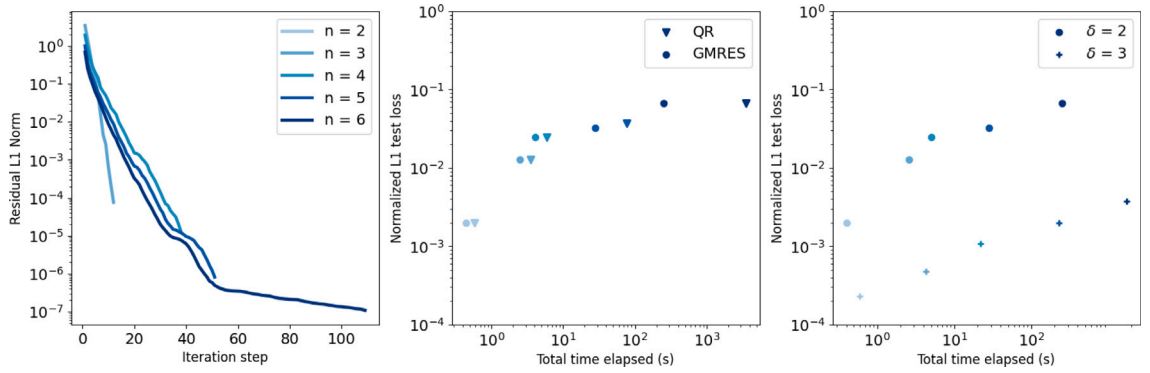
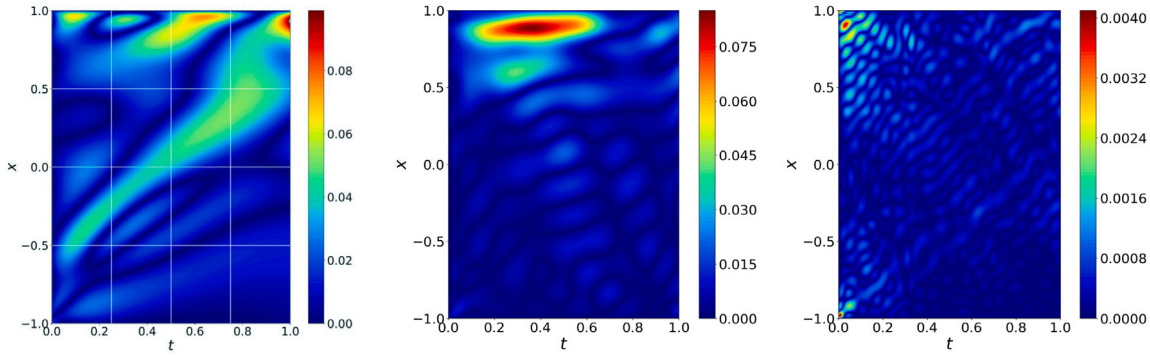


Fig. 10. Weak scaling test for the multi-scale Laplacian problem in Example 5.1. We use $J = 2^n \times 2^n$ subdomains to solve the problem with increasing complexity, where n ranges from 2 to 6. The left figure shows the progression of the residual L^1 norm with each iteration step when using the GMRES method with the AS preconditioner. The middle figure compares the total computation time and test loss for the QR decomposition method and GMRES method with the AS preconditioner. The right figure displays the total computation time for a different overlap ratio with $\delta = 2, 3$.



(a) *hp*-VPINNs (taken from [12]) (b) Penalty method (c) Constraining operator

Fig. 11. Pointwise error of different methods with 4×4 domain decomposition in Example 5.2.

5.3.5. *Comparisons with hp-VPINNs and penalty method*

We consider this equation as a 2-dimensional problem with a 1-dimensional spatial variable and a 1-dimensional temporal variable and solve it in the space–time domain instead of discretizing the temporal variable using the finite difference method. We use the Schwarz domain decomposition with $J = 4 \times 4$ subdomains and compare the results obtained with the *hp*-VPINNs [12]. The overlap ratio δ is fixed as 2. In [12], they employ individual networks in each subdomain with four layers and each with five neurons, resulting in 1296 training parameters, and a 4×4 domain decomposition was adopted. RaNNs with one hidden layer and 81 hidden neurons on each subdomain are used to keep the same total number of parameters, and a uniform distribution $\mathcal{U}(-1, 1)$ is applied to weights and bias parameters. The number of collocation points is set to 160×160, and the threshold parameter τ is fixed by 10^{-3}. The constraining operator is chosen as

$$Cu(x) = L(x)u(x) + G(x), \quad \text{where } L(x) = \tanh(1 + x)\tanh(1 - x)\tanh(t),\ G(x) = -\sin(\pi x), \tag{5.33}$$

such that the boundary condition and initial condition are satisfied.

Fig. 11 illustrates the differences between the exact and numerical solutions obtained using various methods. The reference solution is calculated using a series summation with 800 terms of the solution in [70]. Fig. 11(a) shows results computed using *hp*-VPINNs [12], while Fig. 11(b) presents results using the penalty method to enforce boundary and initial conditions. A penalty parameter of 100 was chosen as optimal from the set {1, 10, 100, 1000}. The computation takes 0.12 s and yields a relative L^2 error of 2.88×10^{-2}. Fig. 11(c) displays the results of applying the constraining operator computed via the CG method with an AS preconditioner, completing in 0.08 s after 138 iterations, and achieving a relative L^2 error of 7.40×10^{-4}, around two orders of magnitude more accurate than *hp*-VPINNs and the penalty method. This demonstrates that our approach not only achieves higher accuracy and efficiency but also satisfies boundary and initial conditions directly, eliminating the need for penalty parameters.

Example 5.3. Given $\Omega = (0, 1)^3$, we consider a three-dimensional problem defined as follows:

$$-\Delta \boldsymbol{u} + \boldsymbol{u} = \boldsymbol{f} \qquad \text{in } \Omega,$$

Table 7
Relative L^2 error and solver time for different methods in Example 5.3.

m	J	DoF	Solver	$Cond$	$Iter$	e_{L^2}	Time(s)
20	$2\times2\times2$	480	QR	10^7	–	1.35e−2	0.41
			CG	10^7	1151	1.63e−2	0.11
			PCG	1	1	1.35e−2	0.001
40	$2\times2\times2$	960	QR	10^{11}	–	1.23e−3	1.04
			CG	10^{11}	1175	2.98e−3	0.43
			PCG	1	3	1.03e−3	0.005
40	$4\times4\times4$	7680	QR	10^{12}	–	8.26e−4	181
			CG	10^{12}	15 236	2.38e−3	385
			PCG	**5.8**	10	8.24e−4	1.41
80	$4\times4\times4$	15360	QR	10^{15}	–	2.37e−5	958
			CG	10^{15}	15 984	3.02e−4	1069
			PCG	**7.4**	11	2.17e−5	5.21

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{g} \qquad \text{on } \partial\Omega.$$

where the exact solution is given by:

$$\boldsymbol{u} = \begin{pmatrix} \cos(2\pi x_1)\sin(2\pi x_2)\sin(2\pi x_3) \\ \sin(2\pi x_1)\cos(2\pi x_2)\sin(2\pi x_3) \\ \sin(2\pi x_1)\sin(2\pi x_2)\cos(2\pi x_3) \end{pmatrix}$$

The Dirichlet boundary condition $\boldsymbol{g}$ and the source term $\boldsymbol{f}$ are derived directly from the exact solution.

5.3.6. Three-dimensional problem

For this problem, we use RaNNs with one hidden layer and three output neurons to approximate $\boldsymbol{u} = (u_1, u_2, u_3)$. The weights and bias parameters of the network are initialized using a uniform distribution $\mathcal{U}(-1, 1)$. The e_{L^2} error is computed approximately on $M = 100\times100\times100$ uniformly spaced test points. The number of collocation points is fixed at 1000 in each subdomain, and the overlap ratio δ is fixed as 2, and the threshold parameter τ is fixed as 10^{-3}.

The constraining operator is chosen as $C\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{L}\boldsymbol{u}(\boldsymbol{x}) + \boldsymbol{G}(\boldsymbol{x})$, where

$$\boldsymbol{L}(\boldsymbol{x}) = \begin{pmatrix} L(\boldsymbol{x}) \\ L(\boldsymbol{x}) \\ L(\boldsymbol{x}) \end{pmatrix}, \ \boldsymbol{G}(x) = \begin{pmatrix} \sin(2\pi x_2)\sin(2\pi x_3) \\ \sin(2\pi x_1)\sin(2\pi x_3) \\ \sin(2\pi x_1)\sin(2\pi x_2) \end{pmatrix}, \ \text{and}$$

$$L(\boldsymbol{x}) = \tanh(x_1)\tanh(1-x_1)\tanh(x_2)\tanh(1-x_2)\tanh(x_3)\tanh(1-x_3).$$

In Table 7, the domain is divided into $J = 2\times2\times2$ and $J = 4\times4\times4$ subdomains, respectively, while the number of hidden neurons m is varied as 20, 40, and 80. We compare the results obtained using the direct least-squares solver via QR decomposition, the CG method without a preconditioner, and the PCG method with the AS preconditioner. The condition numbers of the matrices associated with these methods are listed under *Cond*. The number of iterations for the CG and PCG methods is denoted by $Iter$.

Table 7 illustrates that increasing the number of hidden neurons m and subdomains improves the accuracy for both QR decomposition and PCG method with an AS preconditioner. However, using the CG method without a preconditioner may require a large number of iteration steps and often fails to converge to a satisfactory result. Meanwhile, the computational time required for the PCG method with the AS preconditioner is significantly lower than that for QR decomposition, particularly as the matrix size increases. This highlights the efficiency of the CG method with DD preconditioning in tackling 3-dimensional systems while maintaining accuracy.

As shown on the left of Fig. 12, we display the matrix information for a $4\times4\times4$ domain decomposition with $m = 40$ in Example 5.3. To construct the preconditioner for this equation, we simply need to diagonally stack the preconditioners constructed for $H_i^T H_i$, which corresponds to a scalar problem of the form

$$\begin{aligned} -\Delta u_i + u_i &= f_i \qquad \text{in } \Omega,\ i = 1,2,3, \\ u_i(\boldsymbol{x}) &= g_i \qquad \text{on } \partial\Omega,\ i = 1,2,3. \end{aligned}$$

The resulting global preconditioner retains a block-diagonal structure. As shown in the right plot of Fig. 12, the singular value distribution of the preconditioned system matrix $M_{AS}^{-1/2} H^T H M_{AS}^{-1/2}$ significantly improves. Specifically, the smallest singular value increases from approximately 10^{-6} to 11, while the largest singular value decreases to $8^2 = 64$, bounded by the square of the maximum number of subdomains intersecting at any given collocation point. Consequently, the condition number is dramatically reduced to an exceptionally low value of approximately 5.8.

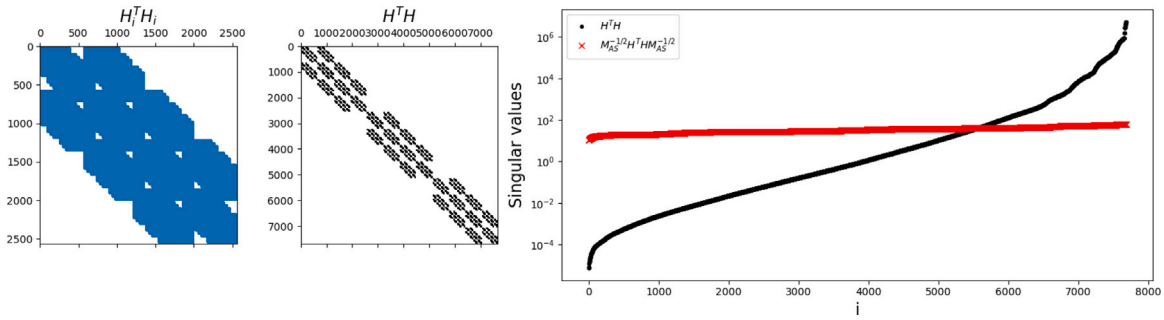


Fig. 12. Matrix structure and singular value distributions for a $4 \times 4 \times 4$ domain decomposition in Example 5.3. The middle and left plots illustrate the structures of the global matrix $H^T H$ and its diagonal block $H_i^T H_i$, respectively. The right plot compares singular value distributions for the preconditioned system using the PCG method with the AS preconditioner and the original unpreconditioned system, arranged in non-decreasing order.

6. Conclusion

In this research, we present a new approach for solving PDEs using randomized neural networks across overlapping subdomains interconnected by local window functions. By integrating a PCA scheme into the neural network structure on the local subdomains, we significantly reduce the number of parameters by eliminating less essential information of neural networks, resulting in a system with a lower condition number. Furthermore, constructing overlapping Schwarz preconditioners for the resulting system enables the efficient solution of the least-squares problem using preconditioned iterative solvers by further reducing the condition number. This means that overlapping Schwarz domain decomposition is combined with RaNNs in two main ways: first, to formulate the least-squares problem, and second, to efficiently construct preconditioners for the resulting linear systems. This is the first paper to demonstrate the application of overlapping Schwarz preconditioners for RaNNs with domain decomposition. The main advantage of this method is the significant reduction in training time compared to traditional neural network training techniques without loss of accuracy.

The integration of RaNNs with DDMs demonstrates considerable potential for addressing multi-scale problems and time-dependent PDEs, offering a robust and efficient solution. However, to solve more complex problems, further investigation into RaNNs combined with multi-level DDM-based architectures is necessary. Specifically, the development of two-level or multi-level Schwarz preconditioners should be pursued. To manage the increased complexity, parallel methods must be developed to ensure computational efficiency. Moreover, careful construction of the corresponding preconditioners is crucial for different approaches using ELMs or RaNNs, particularly when the system includes dense blocks. A valuable direction for future work would be designing a suitable preconditioner for a non-overlapping DDM strategy, such as Local ELMs [31]. Future research should also focus on the numerical analysis of this approach to further refine its theoretical foundation and practical applicability.

CRediT authorship contribution statement

Yong Shang: Writing – original draft, Visualization, Software, Methodology, Data curation. **Alexander Heinlein:** Writing – review & editing, Methodology, Investigation, Formal analysis, Conceptualization. **Siddhartha Mishra:** Writing – review & editing, Supervision, Methodology, Investigation. **Fei Wang:** Writing – review & editing, Supervision, Resources, Project administration, Investigation, Funding acquisition, Conceptualization.

Declaration of competing interest

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

Acknowledgments

We thank the two anonymous reviewers for their valuable suggestions, which have significantly improved this paper.

Data availability

No data was used for the research described in the article.

References

[1] A. Heinlein, A. Klawonn, M. Lanser, J. Weber, Combining machine learning and domain decomposition methods for the solution of partial differential equations—A review, GAMM-Mitt. 44 (1) (2021) e202100001.
[2] A. Klawonn, M. Lanser, J. Weber, Machine learning and domain decomposition methods-a survey, Comput. Sci. Eng. 1 (1) (2024) 2.
[3] M.G. Dissanayake, N. Phan-Thien, Neural-network-based approximations for solving partial differential equations, Commun. Numer. Methods Eng. 10 (3) (1994) 195–201.
[4] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000.
[5] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[6] K.S. McFall, J.R. Mahan, Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions, IEEE Trans. Neural Netw. 20 (8) (2009) 1221–1233.
[7] C. Leake, D. Mortari, Deep theory of functional connections: A new method for estimating the solutions of partial differential equations, Mach. Learn. Knowl. Extr. 2 (1) (2020) 37–55.
[8] L. Lyu, K. Wu, R. Du, J. Chen, Enforcing exact boundary and initial conditions in the deep mixed residual method, 2020, arXiv preprint arXiv:2008.01491.
[9] E. Schiassi, R. Furfaro, C. Leake, M. De Florio, H. Johnston, D. Mortari, Extreme theory of functional connections: A fast physics-informed neural network method for solving ordinary and partial differential equations, Neurocomputing 457 (2021) 334–356.
[10] A.D. Jagtap, E. Kharazmi, G.E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems, Comput. Methods Appl. Mech. Engrg. 365 (2020) 113028.
[11] A.D. Jagtap, G.E. Karniadakis, Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, Commun. Comput. Phys. 28 (5) (2020).
[12] E. Kharazmi, Z. Zhang, G.E. Karniadakis, hp-VPINNs: Variational physics-informed neural networks with domain decomposition, Comput. Methods Appl. Mech. Engrg. 374 (2021) 113547.
[13] W. Li, X. Xiang, Y. Xu, Deep domain decomposition method: Elliptic problems, in: Mathematical and Scientific Machine Learning, PMLR, 2020, pp. 269–286.
[14] H.A. Schwarz, Ueber einen Grenzübergang durch alternirendes Verfahren, Zürcher u. Furrer, 1870.
[15] B. Moseley, A. Markham, T. Nissen-Meyer, Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations, Adv. Comput. Math. 49 (4) (2023) 62.
[16] V. Dolean, A. Heinlein, S. Mishra, B. Moseley, Finite basis physics-informed neural networks as a Schwarz domain decomposition method, in: International Conference on Domain Decomposition Methods, Springer, 2022, pp. 165–172.
[17] V. Dolean, A. Heinlein, S. Mishra, B. Moseley, Multilevel domain decomposition-based architectures for physics-informed neural networks, Comput. Methods Appl. Mech. Engrg. 429 (2024) 117116.
[18] A. Heinlein, A.A. Howard, D. Beecroft, P. Stinis, Multifidelity domain decomposition-based physics-informed neural networks for time-dependent problems, 2024, arXiv preprint arXiv:2401.07888.
[19] A.A. Howard, B. Jacob, S.H. Murphy, A. Heinlein, P. Stinis, Finite basis Kolmogorov–Arnold networks: domain decomposition for data-driven and physics-informed problems, 2024, arXiv preprint arXiv:2406.19662.
[20] S. Anderson, V. Dolean, B. Moseley, J. Pestana, ELM-FBPINN: efficient finite-basis physics-informed neural networks, 2024, arXiv preprint arXiv:2409.01949.
[21] Y.-H. Pao, Y. Takefuji, Functional-link net computing: theory, system architecture, and functionalities, Computer 25 (5) (1992) 76–79.
[22] Y.-H. Pao, G.-H. Park, D.J. Sobajic, Learning and generalization characteristics of the random vector functional-link net, Neurocomputing 6 (2) (1994) 163–180.
[23] B. Igelnik, Y.-H. Pao, Stochastic choice of basis functions in adaptive function approximation and the functional-link net, IEEE Trans. Neural Netw. 6 (6) (1995) 1320–1329.
[24] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1–3) (2006) 489–501.
[25] X. Liu, S. Lin, J. Fang, Z. Xu, Is extreme learning machine feasible? A theoretical assessment (Part I), IEEE Trans. Neural Netw. Learn. Syst. 26 (1) (2014) 7–20.
[26] C. Gallicchio, S. Scardapane, Deep randomized neural networks, in: Recent Trends in Learning from Data: Tutorials from the INNS Big Data and Deep Learning Conference, INNSBDDL2019, Springer, 2020, pp. 43–68.
[27] F. Calabrò, G. Fabiani, C. Siettos, Extreme learning machine collocation for the numerical solution of elliptic PDEs with sharp gradients, Comput. Methods Appl. Mech. Engrg. 387 (2021) 114188.
[28] A. Toselli, O. Widlund, Domain Decomposition Methods-Algorithms and Theory, vol. 34, Springer Science & Business Media, 2006.
[29] B. Smith, P. Bjørstad, W. Gropp, Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge University Press, 1996.
[30] V. Dolean, P. Jolivet, F. Nataf, An Introduction to Domain Decomposition Methods: Algorithms, Theory, and Parallel Implementation, SIAM, 2015.
[31] S. Dong, 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.
[32] S. Dong, Z. Li, A modified batch intrinsic plasticity method for pre-training the random coefficients of extreme learning machines, J. Comput. Phys. 445 (2021) 110585.
[33] S. Dong, N. Ni, A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks, J. Comput. Phys. 435 (2021) 110242.
[34] N. Ni, S. Dong, Numerical computation of partial differential equations by hidden-layer concatenated extreme learning machine, J. Sci. Comput. 95 (2) (2023) 35.
[35] Y. Wang, S. Dong, An extreme learning machine-based method for computational PDEs in higher dimensions, Comput. Methods Appl. Mech. Engrg. 418 (2024) 116578.
[36] J. Sun, S. Dong, F. Wang, Local randomized neural networks with discontinuous Galerkin methods for partial differential equations, J. Comput. Appl. Math. 445 (2024) 115830.
[37] J. Sun, F. Wang, Local randomized neural networks with discontinuous Galerkin methods for diffusive-viscous wave equation, Comput. Math. Appl. 154 (2024) 128–137.
[38] H. Dang, F. Wang, Local randomized neural networks with hybridized discontinuous Petrov–Galerkin methods for Stokes–Darcy flows, Phys. Fluids 36 (8) (2024).
[39] J. Chen, X. Chi, W. E, Z. Yang, Bridging traditional and machine learning-based algorithms for solving PDEs: the random feature method, J. Mach. Learn. 1 (2022) 268–298.
[40] Y. Shang, F. Wang, J. Sun, Randomized neural network with Petrov–Galerkin methods for solving linear and nonlinear partial differential equations, Commun. Nonlinear Sci. Numer. Simul. 127 (2023) 107518.

[41] J. Chen, W. E, Y. Sun, Optimization of random feature method in the high-precision regime, Commun. Appl. Math. Comput. 6 (2) (2024) 1490–1517.
[42] L.F. Pavarino, Domain decomposition algorithms for first-order system least squares methods., ETNA. Electron. Trans. Numer. Anal. [Electron. Only] 8 (1999) 1–14, Publisher: Kent State University, Department of Mathematics and Computer Science.
[43] W. Heinrichs, Least-squares spectral collocation for the Navier–Stokes equations, J. Sci. Comput. 21 (1) (2004) 81–90, http://dx.doi.org/10.1023/B:JOMP.0000027956.13510.5a.
[44] H.-J. Rong, Y.-S. Ong, A.-H. Tan, Z. Zhu, A fast pruned-extreme learning machine for classification problem, Neurocomputing 72 (1–3) (2008) 359–366.
[45] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, A. Lendasse, OP-ELM: optimally pruned extreme learning machine, IEEE Trans. Neural Netw. 21 (1) (2009) 158–162.
[46] G.-B. Huang, L. Chen, Enhanced random search based incremental extreme learning machine, Neurocomputing 71 (16–18) (2008) 3460–3468.
[47] G.-B. Huang, M.-B. Li, L. Chen, C.-K. Siew, Incremental extreme learning machine with fully complex hidden nodes, Neurocomputing 71 (4–6) (2008) 576–583.
[48] A.S. Alencar, A.R.R. Neto, J.P.P. Gomes, A new pruning method for extreme learning machines via genetic algorithms, Appl. Soft Comput. 44 (2016) 101–107.
[49] A.S. Weigend, D.E. Rumelhart, The effective dimension of the space of hidden units, in: [Proceedings] 1991 IEEE International Joint Conference on Neural Networks, IEEE, 1991, pp. 2069–2074.
[50] S. Tamura, M. Tateishi, M. Matumoto, S. Akita, Determination of the number of redundant hidden units in a three-layered feedforward neural network, in: Proceedings of 1993 International Conference on Neural Networks, Vol. 1, IJCNN-93-Nagoya, Japan, IEEE, 1993, pp. 335–338.
[51] M. Hayashi, A fast algorithm for the hidden units in a multilayer perceptron, in: Proceedings of 1993 International Conference on Neural Networks, Vol. 1, IJCNN-93-Nagoya, Japan, IEEE, 1993, pp. 339–342.
[52] J. Xue, J. Li, Y. Gong, Restructuring of deep neural network acoustic models with singular value decomposition, in: Interspeech, 2013, pp. 2365–2369.
[53] D.C. Psichogios, L.H. Ungar, SVD-NET: An algorithm that automatically selects network structure, IEEE Trans. Neural Netw. 5 (3) (1994) 513–515.
[54] S. Abid, F. Fnaiech, M. Najim, A new neural network pruning method based on the singular value decomposition and the weight initialisation, in: 2002 11th European Signal Processing Conference, IEEE, 2002, pp. 1–4.
[55] E.J. Teoh, K.C. Tan, C. Xiang, Estimating the number of hidden neurons in a feedforward network using the singular value decomposition, IEEE Trans. Neural Netw. 17 (6) (2006) 1623–1629.
[56] O.B. Widlund, Iterative substructuring methods: Algorithms and theory for elliptic problems in the plane, in: First International Symposium on Domain Decomposition Methods for Partial Differential Equations, Philadelphia, PA, 1988, pp. 113–128.
[57] H. Abdi, L.J. Williams, Principal component analysis, Wiley Interdiscip. Rev.: Comput. Stat. 2 (4) (2010) 433–459.
[58] D.G. Luenberger, Introduction to Linear and Nonlinear Programming, vol. 28, Addison-Wesley Reading, MA, 1973.
[59] T.F. Chan, T.P. Mathew, Domain decomposition algorithms, Acta Numer. 3 (1994) 61–143.
[60] X.-C. Cai, M. Sarkis, A restricted additive Schwarz preconditioner for general sparse linear systems, SIAM J. Sci. Comput. 21 (2) (1999) 792–797.
[61] Y. Saad, Iterative Methods for Sparse Linear Systems, SIAM, 2003.
[62] T. Luo, Z.-Q.J. Xu, Z. Ma, Y. Zhang, Phase diagram for two-layer ReLU neural networks at infinite-width limit, J. Mach. Learn. Res. 22 (71) (2021) 1–47.
[63] H. Zhou, Z. Qixuan, T. Luo, Y. Zhang, Z.-Q. Xu, Towards understanding the condensation of neural networks at initial training, Adv. Neural Inf. Process. Syst. 35 (2022) 2184–2196.
[64] H. Dang, F. Wang, S. Jiang, Adaptive growing randomized neural networks for solving partial differential equations, 2024, arXiv preprint arXiv:2408.17225.
[65] Z. Zhang, F. Bao, L. Ju, G. Zhang, Transferable neural networks for partial differential equations, J. Sci. Comput. 99 (1) (2024) 2.
[66] D.E. De Falco, F. Calabrò, M. Pragliola, Insights on the different convergences in extreme learning machine, Neurocomputing 599 (2024) 128061.
[67] Y. Shang, F. Wang, Randomized neural networks with Petrov–Galerkin methods for solving linear elasticity and Navier–Stokes equations, J. Eng. Mech. 150 (4) (2024) 04024010.
[68] T. De Ryck, S. Mishra, Y. Shang, F. Wang, Approximation theory and applications of randomized neural networks for solving high-dimensional PDEs, 2025, arXiv preprint arXiv:2501.12145.
[69] G. Fabiani, F. Calabrò, L. Russo, C. Siettos, Numerical solution and bifurcation analysis of nonlinear partial differential equations with extreme learning machines, J. Sci. Comput. 89 (2) (2021) 44.
[70] A. Mojtabi, M.O. Deville, One-dimensional linear advection–diffusion equation: Analytical and finite element solutions, Comput. & Fluids 107 (2015) 189–195.