Neural Processing Letters (2026) 58:49
https://doi.org/10.1007/s11063-026-11853-2

Revolutionizing High-Dimensional PDE Solutions: A Novel Approach of Integrated Truncated Singular Value Decomposition to Enhance Physics-Informed Neural Networks

Danang A. Pratama[1] · Maharani A. Bakar[2] · Sugiarto Surono[3] · Abdellah Salhi[4] · Nur Baini Ismail [2] · Norizan Mohamed[1]

Received: 29 June 2024 / Accepted: 20 April 2026 / Published online: 5 June 2026

Abstract
Partial differential equations (PDEs) serve as crucial tools for modeling complex real-world phenomena. The physics-informed neural network (PINN) is an advanced method within artificial neural network (ANN)-based approaches for solving a wide range of PDE problems. The restarting strategy of PINN (r-PINN) is a novel modification that has demonstrated a significant reduction in computational time compared to its predecessor. However, the complexity of the r-PINN architecture poses challenges, particularly in accurately solving PDE problems with complex geometries, resulting in a large number of trainable parameters that affect computational performance and memory requirements. To address these challenges, this study integrates truncated singular value decomposition (TSVD) into both the PINN and r-PINN frameworks to uncover the underlying structure of the trainable parameter matrices, enabling their factorization into low-rank components. This process yields a compressed trainable parameter matrix, improving training efficiency while preserving accu-

✉ Maharani A. Bakar
maharani@umt.edu.my

Danang A. Pratama
p5564@pps.umt.edu.my

Sugiarto Surono
sugiyarto@math.uad.ac.id

Abdellah Salhi
as@essex.ac.uk

Nur Baini Ismail
baini@umt.edu.my

Norizan Mohamed
norizan@umt.edu.my

1 Faculty of Computer Science and Mathematics, Universiti Malaysia Terengganu, Kuala Nerus, Terengganu 21030, Malaysia

2 Special Interest Group Modelling and Data Analytics, Universiti Malaysia Terengganu, Kuala Nerus, Terengganu 21030, Malaysia

3 Mathematics Department, Faculty of Applied Sciences and Technology, Universitas Ahmad Dahlan, Yogyakarta, Indonesia

4 Department of Mathematical Sciences, University of Essex, Wivenhoe Park, CO43SQ Colchester, UK

racy. The proposed approach aims to obtain numerical solutions for various benchmark PDE problems, including up to three-dimensional cases and a PDE system. This approach is applicable not only to the novel r-PINN but also to the basic PINN, allowing for an analysis of the improvements gained through TSVD. Comparative analyses with non-TSVD approaches will be presented, and experimental results will demonstrate the efficacy of TSVD in enhancing computational performance across all benchmark problems, warranting further investigation.

Keywords Partial differential equations · PINN · r-PINN · TSVD · Low-rank matrices

1 Introduction

Partial differential equations (PDEs) are equations that describe the relationship between an unknown function of two or more independent variables and their partial derivatives [1, 2]. Many different kinds of PDEs are used to analyze processes distributed in space and time. Specifically, PDEs can model and analyze complex real-world phenomena such as fluid dynamics, heat transfer, electromagnetic fields, quantum mechanics, and more. One of the most well-known PDEs, the heat equation, can estimate the spread of thermal energy within materials [3]. The wave equation is also able to describe time-dependent displacement in an elastic material [4]. Laplace's equation is widely applied in electrostatics, fluid flow, geometry, and the theory of harmonic functions [5]. Lastly, the vulnerability of ecological environments can also be analyzed using PDEs [6]. As elegant as these models are, studying PDEs remains an active research field of considerable interest.

The solution to a PDE can be obtained either analytically or numerically. However, obtaining an analytical solution can be challenging or even impossible for complex problems, either because a solution has not yet been found or because it cannot be expressed using elementary functions [7, 8]. As an alternative, numerical methods are used to approximate the solution. Classical numerical methods such as the Finite Difference Method (FDM), Finite Element Method (FEM), and Runge-Kutta have emerged in recent years as tools to obtain numerical solutions to PDEs. However, these methods rely on domain discretization and become computationally expensive, especially when dealing with high-dimensional problems or complex geometries [9, 10]. Furthermore, these classical methods are also inappropriate for real-time use and are time-consuming to adapt to new problems, particularly for non-experts in numerical mathematics and computational modeling [11]. Recently, artificial neural networks (ANNs) have emerged as a new class of approaches for obtaining numerical solutions to PDEs.

Artificial Neural Networks (ANNs) are inspired by the structure and function of biological neural networks. They have gained significant attention over the past two decades [12]. In recent years, ANNs have shown remarkable success in handling various tasks such as natural language processing [13], image classification [14], and error detection [15]. ANNs also have tremendous advantages in function approximation due to their role as universal function approximators [16]. These advantages make ANNs a promising approach for obtaining numerical solutions to PDEs and offer the potential to achieve high accuracy and efficiency.

Previous studies have demonstrated the potential of ANNs to solve various differential equations (DEs). Lee and Kang [17] solved first-order ordinary differential equations (ODEs) by implementing a Hopfield neural network model using a parallel architecture. As a result, their parallel neural algorithm is applicable for solving a wide range of DE problems. Lagaris et al. [18] used different ANN techniques to solve DE problems by incorporating a trial

solution into their methods. This trial solution technique succeeded in providing accurate solutions for both ODEs and PDEs.

ANNs also offer flexibility, allowing them to be combined with other techniques to enhance results. Aarts and Van der Veer demonstrated the use of ANN to solve a series of linear and nonlinear DEs [19, 20]. Their ANN model was trained simultaneously with an evolutionary algorithm to obtain more efficient results. Malek and Beidokhti combined ANN with the Nelder-Mead simplex method to find numerical solutions for higher-order PDEs [21]. This hybrid method provided excellent solutions for both lower- and higher-order PDE problems. Several other studies have explored the integration of ANNs with different computational methods to improve performance and accuracy. For instance, Sirignano and Spiliopoulos [22] combined ANNs with the Galerkin method to form the Deep Galerkin Method. Khan et al. [23] applied a hybrid of ANNs with the Generalized Normal Distribution Optimization (GNDO) and the Interior Point Algorithm (IPA). In another study, ANNs were integrated with the meta-heuristic Biogeography-Based Heterogeneous Cuckoo Search (BHCS) to enhance numerical efficiency [24]. Additionally, Khan et al. [25] employed a hybrid model that combines ANNs with the Whale Optimization Algorithm (WOA) and the Nelder-Mead (NM) method. These examples reflect the growing trend of hybridizing ANNs with advanced optimization or numerical techniques to solve increasingly complex differential equation problems.

The Physics-Informed Neural Network (PINN) is one of the most recent ANN-based approaches for solving PDEs and has gained significant attention in recent years [26–28]. However, it has been observed to have certain drawbacks, particularly the high training cost, which can adversely affect both performance and accuracy. To improve accuracy, especially when dealing with PDEs involving complex geometries, PINNs often require increased architectural complexity. This typically involves more neurons, deeper layers, and results in a large number of trainable parameters (weights and biases), which in turn increases computational time and resource consumption. Therefore, compressing the trainable parameters while maintaining accuracy is essential. Low-rank matrix factorization (MF) is one of the most commonly used ANN compression techniques to date. This approach is popular because it has been proven to reduce the number of trainable parameters, shorten training time, and enhance interpretability in many applications.

This study implements one of the matrix factorization (MF) techniques, known as truncated singular value decomposition (TSVD), to be integrated with both PINN and r-PINN methods for solving several benchmark PDE problems. TSVD is a variant of singular value decomposition (SVD) in which the number of singular values is limited by a predefined rank variable. It is a popular alternative to standard SVD due to its improved computational efficiency and interpretability. TSVD is incorporated within the ANN layers to uncover the latent structure of the trainable parameter matrix connecting each layer, factoring it into low-rank matrices to provide a compressed representation. This compressed matrix significantly reduces the number of trainable parameters and accelerates the training process. This technique has previously been applied by Cai et al. [29] in image classification tasks, where it achieved higher accuracy and compression ratios. A comparison between the original PINN and the novel r-PINN, both with and without TSVD integration, will be conducted to determine whether integrating TSVD is an effective way to enhance performance. The key contributions of this study are summarized as follows:

1. TSVD is constructed within the architecture of ANNs for both basic PINN and r-PINN methods and applied to approximate solutions for several benchmark PDE problems, including those up to three dimensions and systems of PDEs.

2. A loss function based on mean square error is formulated and monitored to update the trainable parameters, which consist of weights and biases, using the Adam optimization algorithm.
3. Several proposed actions are outlined based on the findings and the observed impact of variations in the proposed approach.
4. The approximate results obtained using the integrated TSVD method are compared with those from the non-TSVD approach and analytical solutions. The comparison includes the absolute error distribution and mean absolute error (MAE) to validate the accuracy and effectiveness of the proposed mechanism.
5. The approximate solutions, trainable parameters, computational time, and loss performance for both the TSVD and non-TSVD approaches are presented using graphs and tables, demonstrating the effectiveness and robustness of the proposed method.

Section 2 provides an overview of the PINN and r-PINN workflows for solving PDE problems, along with a review of related work on TSVD as one of the MF techniques. Section 3 outlines the numerical implementation, detailing how TSVD is integrated into the proposed approach to solve the general form of PDE problems. Section 4 presents the implementation results of the integrated TSVD approaches in solving PDEs involving up to three spatial dimensions and one PDE system, including comparisons with the non-TSVD approaches. Finally, Section 5 concludes the study by summarizing the key findings.

2 Literature Review

2.1 Physic-Informed Neural Network

The Physics-Informed Neural Network (PINN) is an efficient ANN-based method for solving unsupervised learning tasks by enforcing physical laws specified by partial differential equations (PDEs), which are incorporated as regularization terms in the loss function [27]. Currently, PINNs are receiving significant attention from researchers and are being applied across various fields, such as learning advection–diffusion–reaction (ADR) systems [30], parameter estimation in subsurface problems [31, 32], predicting blood pressure [33], and solving nonlinear multiphysics problems [34], among others.

The basic architecture of PINNs often struggles with large network size and excessive parameters. Common architectures such as dense networks, convolutional neural networks [35], and recurrent neural networks [36] typically involve deep and wide layers, resulting in long training times, high memory usage, and increased computational cost, especially in high-dimensional or real-time applications. To mitigate these challenges, researchers have explored model compression using low-rank factorization methods such as Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF) to compress models while maintaining accuracy [37, 38]. For instance, Greenformers applied SVD-based compression to convolutional layers, achieving up to 58.97% efficiency improvement with no performance loss [39]. Truncated SVD (TSVD), which retains only the most significant singular values, has also shown strong potential in ANN-based image classification tasks [29].

Recent studies have explored the scalability of PINNs by applying tensor decomposition methods, including CP, Tensor Train, and Tucker, which enable efficient variable separation and reduce model size [40]. A pruning strategy has also been proposed to accelerate convergence in enriched PINNs for two-phase flow in heterogeneous porous media, thereby

improving efficiency while maintaining accuracy [41]. Collectively, these advances in compressing PINN complexity highlight a growing diversity of techniques aimed at addressing limitations in efficiency and scalability for solving a wide range of PDEs.

2.2 Restarting PINN for Solving Partial Differential Equation

The original PINN usually used fixed iteration and frequently had limitations The basic PINN typically uses a fixed number of iterations and often suffers from limitations that cause it to run through all iterations without significantly reducing the loss function value, thereby wasting computational time [42]. To address this issue, the r-PINN method employs a restarting scheme adopted from [43], performing only a small number of iterations as a modification to the original PINN. To better understand the idea behind this approach, consider the following general high-dimensional d-th order PDE [44]:

$$\mathcal{F}\left(x, y, t, u, u_x, u_y, u_t, \ldots, u_{x^{(d)}}, u_{y^{(d)}}, u_{t^{(d)}}\right) = f(x, y, t), \ (x, y) \in \Omega, t \in [0, T]. \quad (1)$$

This equation is defined over the domain $\Omega \subset \mathbb{R}^2$, subject to the following initial and boundary conditions:

$$\begin{aligned} u(x_0, y_0, 0) &= u_0, \ (x_0, y_0) \in \partial\Omega, \\ u(x_b, y_b, t) &= g(x_b, y_b, t), \ (x_b, y_b) \in \partial\Omega, t \in [0, T]. \end{aligned} \quad (2)$$

Similar to the original PINN, r-PINN aims to obtain a trained ANN function, $\hat{u}(x, y, t; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a matrix of trainable parameters consisting of weights and biases, to approximate an actual unknown function $u(x, y, t)$. The ANN architecture includes three inputs (x, y, and t), m hidden layers with n neurons each, and a single output that produces $\hat{u}(x, y, t; \boldsymbol{\theta})$. A loss function based on mean square error (MSE) is constructed and monitored using the Adam optimizer [45] to update the trainable parameters. The Adam optimizer is chosen because it has shown superior performance in optimizing r-PINN compared to other gradient-based optimizers, as reported in [42].

Furthermore, the loss value is monitored throughout the training process, and if no further improvement is observed over several upcoming iterations, the process is terminated and restarted from the beginning. The best trainable parameters obtained during each cycle are saved and used in the subsequent step until a predefined number of cycles is completed. The complete scheme of the r-PINN method is illustrated in Figure 1.

2.3 Low-rank Matrix Factorization

The weight and bias parameters that connect each layer of an ANN typically take the form of arrays or matrices. This has made low-rank matrix factorization (MF) one of the most widely used ANN compression techniques to date. The key idea of MF is to uncover the latent structure in these matrices by factoring them into low-rank components, providing a compressed representation [46]. Previous research suggests that integrating MF with ANN is a promising approach for reducing model size and improving efficiency and performance. Flenner and Hunter [47] introduced a deep non-negative matrix factorization (NMF) framework by combining multiple layers of NMF with pooling layers in a convolutional model trained on limited text and audio data, outperforming deep neural networks that typically require large amounts of training data. Guo et al. [48] proposed a sparse deep NMF model that incorporated Nesterov's accelerated algorithm to speed up the computation process during

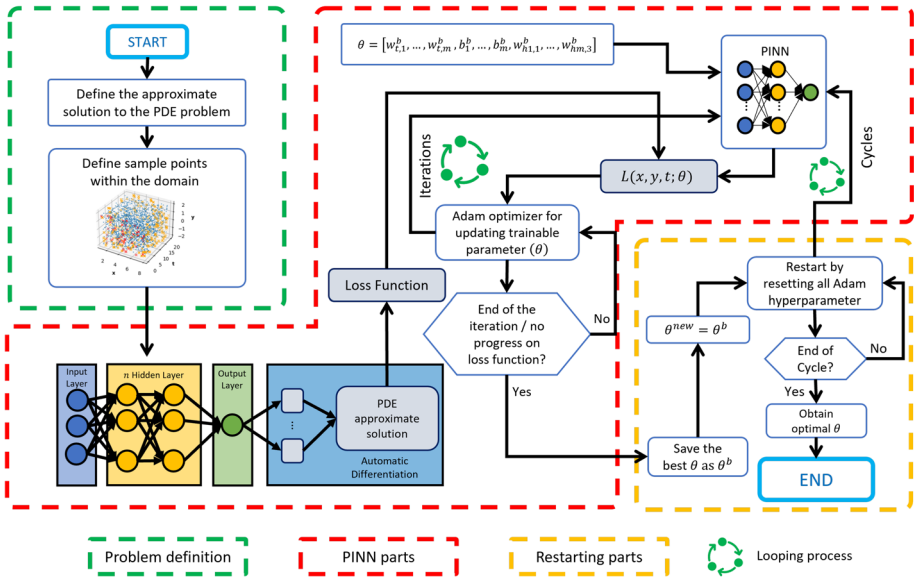


Fig. 1 The scheme of r-PINN for solving PDE

image classification. These studies highlight the potential of matrix factorization techniques to enhance ANN efficiency and performance.

2.4 Truncated Singular Value Decomposition

Singular Value Decomposition (SVD) is a fundamental technique among MF methods and is widely used in real-world systems and powerful data processing applications in the computational domain [49]. In SVD, a rectangular matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ (where $m \geq n$) can be represented as the product of three matrices: the left singular matrix $\mathbf{U} \in \mathbb{R}^{m \times n}$, the diagonal singular matrix $\Sigma \in \mathbb{R}^{n \times n}$, and the right singular matrix $\mathbf{V^T} \in \mathbb{R}^{n \times n}$, which can be written as [50]:

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V^T}, \tag{3}$$

where $\mathbf{U}$ and $\mathbf{V^T}$ are orthogonal, with column vectors $u_i, i = 1, \ldots, m$, and $v_j, j = 1, \ldots, n$, respectively. Furthermore, the diagonal entries σ_j of Σ are called singular values and are ordered as $\sigma_1 > \sigma_2 > \ldots > \sigma_n \geq 0$ [51]. However, instead of performing the full SVD factorization, the decomposition can be approximated by truncating the top r singular values from the three matrices, a method known as TSVD. In other words, matrix $\mathbf{A}$ can be approximated using the largest r singular values, where $r \leq \min(m, n)$. Therefore, the approximated matrix Θ can be constructed as follows:

$$\mathbf{A} \approx \Theta = \mathbf{U}_r \Sigma_r \mathbf{V^T}_r, \tag{4}$$

where $\mathbf{U}_r \in \mathbb{R}^{m \times r}$, $\Sigma_r \in \mathbb{R}^{r \times r}$, and $\mathbf{V^T}_r \in \mathbb{R}^{r \times n}$. The objective of TSVD is to minimize the Frobenius norm between the original matrix $\mathbf{A}$ and the approximated matrix Θ, as expressed by the following:

$$min \|\mathbf{A} - \mathbf{U}_r \Sigma_r \mathbf{V^T}_r\|_F^2. \tag{5}$$

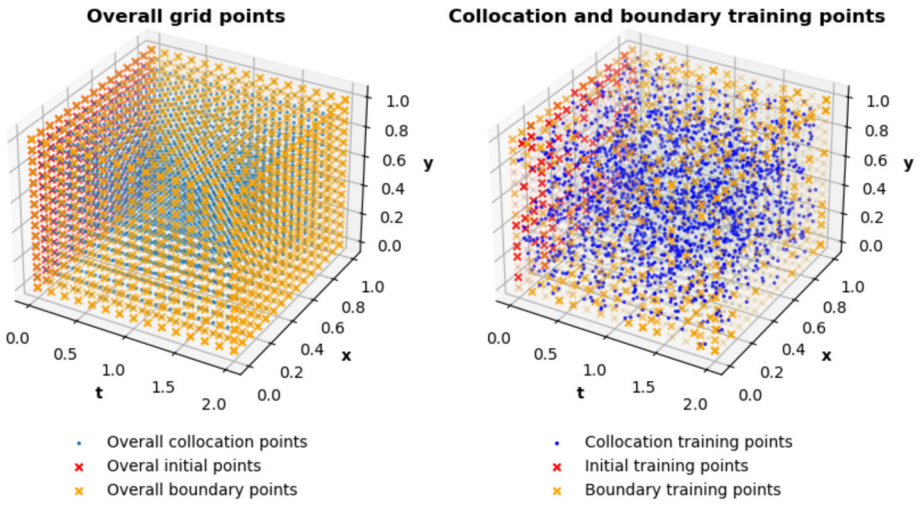


Fig. 2 Collocation, initial, and boundary sample points selected randomly within the PDE domain

The approximated matrix Θ lends itself to a matrix approximation scheme that is computationally cheaper than the full SVD.

3 r-PINN Integrated with TSVD for Solving PDE Problems

TSVD has not yet been integrated into the PINN framework for solving PDEs, highlighting a key research gap. Given ongoing challenges with long training times and model complexity, this section discusses a TSVD-based compression layer that reduces the number of trainable parameters while maintaining accuracy. This approach is expected to significantly enhance computational efficiency, making PINNs more practical and scalable for solving DEs in real-world applications.

3.1 Define the Sample of PDE Collocation Points and Initial/boundary Points

After the PDE is defined as in Equation 1, the next step is to define the points inside the PDE domain. The domain of a PDE typically refers to the grid points over the intervals of the independent variables that describe the solution, usually represented within a rectangular area. Using a random normal distribution, a small portion of sample points is selected to represent the overall domain for ANN training. These points are subsequently categorized into collocation points ($N_{\mathcal{F}}$) and initial or boundary points (N_0 or N_b), respectively. A simple illustration of the sample and overall points is shown in Figure 2.

3.2 Define the ANN Architecture

Generally, the architecture of an ANN consists of multiple layers composed of many units (neurons), including input, hidden, and output layers [52]. In this case, the input layer consists of three neurons corresponding to the PDE's independent variables (x_i, y_j, t_k), which are

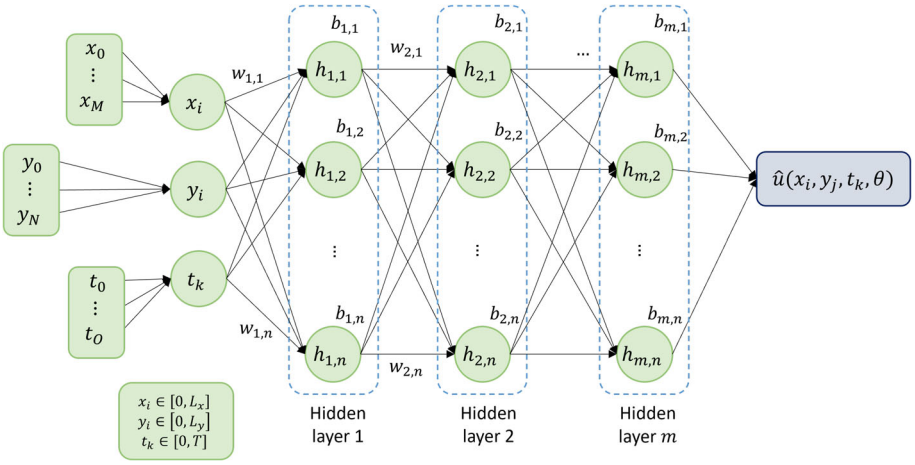


Fig. 3 ANN architecture containing m layers with n units (neurons)

derived from the previously established sample points and fed sequentially into the ANN model.

In ANN terminology, the hidden layer is located between the input and output layers and sets up their connections. Each neuron in the hidden layer has a bias that shifts its activation toward the positive or negative direction. All neurons are connected by a value called a weight, which is initialized randomly based on a chosen initializer [53–55]. The weights and biases are stored as trainable parameters denoted by $\boldsymbol{\theta}$. Lastly, the output layer has a single output $\hat{u}(x_i, y_j, t_k; \boldsymbol{\theta})$ that approximates the unknown PDE function. See Figure 3 for a detailed visualization of the architecture.

3.3 r-PINN with TSVD For Solving PDE

As mentioned previously, the r-PINN approach to solving PDEs begins by initializing the trainable parameters $\boldsymbol{\theta}$. In this case, the He normal distribution is employed [53]. A fully connected (dense) layer is used as the first hidden layer, and its formulation is shown in Eq. 6.

$$\begin{bmatrix} h_{1,1} \\ h_{1,2} \\ \vdots \\ h_{1,n} \end{bmatrix}_{n\times 1} = \begin{bmatrix} w_{1,1} & w_{1,n+1} & w_{1,2n+1} \\ w_{1,2} & w_{1,n+2} & w_{1,2n+2} \\ \vdots & \vdots & \vdots \\ w_{1,n} & w_{1,2n} & w_{1,3n} \end{bmatrix}_{n\times 3} \begin{bmatrix} x_i \\ y_i \\ t_k \end{bmatrix}_{3\times 1} + \begin{bmatrix} b_{1,1} \\ b_{1,2} \\ \vdots \\ b_{1,n} \end{bmatrix}_{n\times 1}, \tag{6}$$

The trainable parameters $\boldsymbol{\theta}$ are then used to form a linear combination with the inputs (x_i, y_j, t_k), where $i = 0,\ldots, M$, $j = 0,\ldots, N$, and $k = 0,\ldots, O$, respectively. The resulting terms are arranged into an operation matrix and processed as follows:

$$\begin{bmatrix} h_{1,1} \\ h_{1,2} \\ \vdots \\ h_{1,n} \end{bmatrix}_{n\times 1} = \begin{bmatrix} \left(w_{1,1}x_i + w_{1,n+1}y_j + w_{1,2n+1}t_k\right) + b_{1,1} \\ \left(w_{1,2}x_i + w_{1,n+2}y_j + w_{1,2n+1}t_k\right) + b_{1,2} \\ \vdots \\ \left(w_{1,n}x_i + w_{1,2n}y_j + w_{1,3n}t_k\right) + b_{1,n} \end{bmatrix}_{n\times 1}. \tag{7}$$

The activation function in the hidden layer plays a crucial role in the learning performance of ANNs [56]. The hyperbolic tangent (*tanh*) activation function is employed and defined as follows:

$$f(h_{1,p}) = \frac{e^{h_{1,p}} - e^{-h_{1,p}}}{e^{h_{1,p}} + e^{-h_{1,p}}}. \tag{8}$$

This activation function is used as it has been empirically shown to offer superior approximation performance compared to other activation functions [57]. This is applied elementwise to $h_{1,p}$, where $p = 1, \ldots, n$, and the resulting values serve as the input to the next layer.

The next step is the process from the current hidden layer to the next hidden layer, or from the next hidden layer to the subsequent layer, until the last hidden layer. Equations 9 and 10 demonstrate the transformation from the current hidden layer h_l to the next hidden layer h_{l+1} ($l = 1, \ldots, m$).

$$\begin{bmatrix} h_{l+1,1} \\ h_{l+1,2} \\ \vdots \\ h_{l+1,n} \end{bmatrix}_{n\times 1} = \begin{bmatrix} w_{l,1} & w_{l,n+1} & \cdots & w_{l,n(n-1)+1} \\ w_{l,2} & w_{l,n+2} & \cdots & w_{l,n(n-1)+2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{l,n} & w_{l,2n} & \cdots & w_{l,nn} \end{bmatrix}_{n\times n} \begin{bmatrix} f(h_{l,1}) \\ f(h_{l,2}) \\ \vdots \\ f(h_{l,n}) \end{bmatrix}_{n\times 1} + \begin{bmatrix} b_{l+1,1} \\ b_{l+1,2} \\ \vdots \\ b_{l+1,n} \end{bmatrix}_{n\times 1}, \tag{9}$$

The calculation process is similar to the previous step, and the result is shown below:

$$\begin{bmatrix} h_{l+1,1} \\ h_{l+1,2} \\ \vdots \\ h_{l+1,n} \end{bmatrix}_{n\times 1} = \begin{bmatrix} \left(w_{l,1} f(h_{l,1}) + w_{l,n+1} f(h_{l,2}) + \ldots + w_{l,n(n-1)+1} f(h_{l,n})\right) + b_{l+1,1} \\ \left(w_{l,2} f(h_{l,1}) + w_{l,n+2} f(h_{l,2}) + \ldots + w_{l,n(n-1)+2} f(h_{l,n})\right) + b_{l+1,2} \\ \vdots \\ \left(w_{l,n} f(h_{l,1}) + w_{l,2n} f(h_{l,2}) + \ldots + w_{l,nn} f(h_{l,n})\right) + b_{l+1,n} \end{bmatrix}_{n\times 1}. \tag{10}$$

It can be seen that the weight matrix in Equations 9 (say, $\mathbf{W}_l$) has $n \times n$ elements. TSVD is used to decompose $\mathbf{W}_l$ so that the total number of weight parameters is reduced. The decomposition process of $\mathbf{W}_l$ can be written as:

$$\begin{bmatrix} w_{l,1} & w_{l,n+1} & \cdots & w_{l,n(n-1)+1} \\ w_{l,2} & w_{l,n+2} & \cdots & w_{l,n(n-1)+2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{l,n} & w_{l,2n} & \cdots & w_{l,nn} \end{bmatrix}_{n\times n} \approx \begin{bmatrix} u^l_{1,1} & \cdots & u^l_{1,r} \\ u^l_{2,1} & \ddots & u^l_{2,r} \\ \vdots & \vdots & \vdots \\ u^l_{n,1} & \cdots & u^l_{n,r} \end{bmatrix}_{n\times r} \begin{bmatrix} \sigma^l_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma^l_r \end{bmatrix}_{r\times r} \begin{bmatrix} v^l_{1,1} & v^l_{1,2} & \cdots & v^l_{1,n} \\ \vdots & \ddots & \vdots & \vdots \\ v^l_{r,1} & v^l_{r,2} & \cdots & v^l_{r,n} \end{bmatrix}_{r\times n} \tag{11}$$

Equation 11 shows the decomposition process of $\mathbf{W}_l$ into the product of three matrices: $\mathbf{U}^l_r$, Σ^l_r, and $\mathbf{V}^{lT}_r$. These matrices are then truncated to retain only the top r components, as explained in Section 2.4. The process reduces the number of weight parameters compared to the original weight matrix $\mathbf{W}l$, so that $h_{l+1,k}$ can be obtained as follows:

$$\begin{bmatrix} h_{l+1,1} \\ h_{l+1,2} \\ \vdots \\ h_{l+1,n} \end{bmatrix}_{n\times 1} = \begin{bmatrix} \left(\sum_{i=1}^{r} u_{1i}\sigma_i v_{i1} f(h_{l,1}) + \sum_{i=1}^{r} u_{1i}\sigma_i v_{i2} f(h_{l,2}) + \ldots + \sum_{i=1}^{r} u_{1i}\sigma_i v_{in} f(h_{l,n})\right) + b_{l+1,1} \\ \left(\sum_{i=1}^{r} u_{2i}\sigma_i v_{i1} f(h_{l,1}) + \sum_{i=1}^{r} u_{2i}\sigma_i v_{i2} f(h_{l,2}) + \ldots + \sum_{i=1}^{r} u_{2i}\sigma_i v_{in} f(h_{l,n})\right) + b_{l+1,2} \\ \vdots \\ \left(\sum_{i=1}^{r} u_{ni}\sigma_i v_{i1} f(h_{l,1}) + \sum_{i=1}^{r} u_{ni}\sigma_i v_{i2} f(h_{l,2}) + \ldots + \sum_{i=1}^{r} u_{ni}\sigma_i v_{in} f(h_{l,n})\right) + b_{l+1,n} \end{bmatrix}_{n\times 1} \tag{12}$$

The final step involves computing outputs to obtain $\hat{u}(x_i, y_j, t_k; \boldsymbol{\theta})$ from the last hidden layer to the output layer. The operations matrix in Equation 13 specifies the process.

$$\hat{u}(x_i, y_j, t_k; \boldsymbol{\theta}) = \begin{bmatrix} w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix}_{1\times n} \begin{bmatrix} f(h_{m,1}) \\ f(h_{m,2}) \\ \vdots \\ f(h_{m,n}) \end{bmatrix}_{n\times 1} + \begin{bmatrix} b_u \end{bmatrix}_{1\times 1}, \tag{13}$$

such that

$$\begin{aligned} \hat{u}(x_i, y_j, t_k; \boldsymbol{\theta}) &= \left(w_{m,1} f(h_{m,1}) + w_{m,2} f(h_{m,2}) + \ldots + w_{m,n} f(h_{m,n}) \right) + b_u \\ &= \sum_{i=1}^{n} w_{m,i} f(h_{m,i}) + b_u. \end{aligned} \tag{14}$$

The output $\hat{u}(x_i, y_j, t_k; \boldsymbol{\theta})$ represent the network's approximation to the unknown function $u(x_i, y_j, t_k)$, which is governed by the PDE in Equation 1. Evaluating this output is crucial for assessing how well the network satisfies the physical constraints imposed by the PDE.

Once the output is obtained, the next step is to compute the d-th derivatives of $\hat{u}(x_i, y_j, t_k; \boldsymbol{\theta})$. These derivative are evaluated using automatic differentiation (AD), which is implemented in TensorFlow [58], as shown in the following equations.

$$\begin{aligned} \hat{u}_{x^{(d)}}(x_i, y_j, t_k; \boldsymbol{\theta}) &= \frac{\partial^{(d)}}{\partial x_i^{(d)}} \sum_{i=1}^{n} w_{m,i} f(h_{m,i}), \\ \hat{u}_{y^{(d)}}(x_i, y_j, t_k; \boldsymbol{\theta}) &= \frac{\partial^{(d)}}{\partial y_j^{(d)}} \sum_{i=1}^{n} w_{m,i} f(h_{m,i}), \\ \hat{u}_{t^{(d)}}(x_i, y_j, t_k; \boldsymbol{\theta}) &= \frac{\partial^{(d)}}{\partial t_k^{(d)}} \sum_{i=1}^{n} w_{m,i} f(h_{m,i}). \end{aligned} \tag{15}$$

Collectively, this process establishes the forward pass and the derivative evaluation required to formulate the PDE residuals before constructing the loss function used for training.

3.4 Build the Loss Function

After obtaining the approximate output $\hat{u}(x_i, y_j, t_k; \boldsymbol{\theta})$ and its derivatives, the next main task is to define the loss function. The loss function of PINN is expressed as follows:

$$\mathcal{L}(\boldsymbol{\theta}) = MSE_0 + MSE_b + MSE_{\mathcal{F}}, \tag{16}$$

where MSE_0, MSE_b, and $MSE_{\mathcal{F}}$ are mean squared error (MSE)-based loss functions corresponding to the initial conditions, boundary conditions, and the governing PDE, as given in Equation 17.

$$\begin{cases} MSE_0 = \frac{1}{N_0}\sum_{i=1}^{N_0}\left[\hat{u}(x_0^i, t_0^i; \boldsymbol{\theta}) - u_0(x_0^i, y_0^i, t_0^i)\right]^2, \\ MSE_b = \frac{1}{N_b}\sum_{i=1}^{N_b}\left[\hat{u}(x_b^i, t_b^i; \boldsymbol{\theta}) - g(x_b^i, y_b^i, t_b^i)\right]^2, \\ MSE_{\mathcal{F}} = \frac{1}{N_{\mathcal{F}}}\sum_{i=1}^{N_{\mathcal{F}}}\left[\mathcal{F}\left(x_{\mathcal{F}}^i, y_{\mathcal{F}}^i, t_{\mathcal{F}}^i, \hat{u}^i, \hat{u}_x^i, \hat{u}_y^i, \hat{u}_t^i, \ldots, \hat{u}^i_{x^{(d)}}, \hat{u}^i_{y^{(d)}}, \hat{u}^i_{t^{(d)}}\right) - f(x_{\mathcal{F}}^i, y_{\mathcal{F}}^i, t_{\mathcal{F}}^i)\right]^2. \end{cases} \tag{17}$$

where,

(x_0^i, y_0^i, t_0^i), and (x_b^i, y_b^i, t_b^i): Randomly distributed initial points and boundary points, respectively.

$(x_{\mathcal{F}}^i, y_{\mathcal{F}}^i, t_{\mathcal{F}}^i)$: Randomly distributed collocation points in the domain.

N_0, N_b and $N_{\mathcal{F}}$: Total number of sample data for initial points, boundary points, and collocation points, respectively.

The most effective approach to achieving an optimal value of the loss function is to minimize it toward zero. This is accomplished using the Adam optimizer, which iteratively adjusts the values of $\boldsymbol{\theta}$ to converge toward the optimal solution.

3.5 Adam Optimizer

The trainable parameter $\boldsymbol{\theta}$ is updated to minimize the loss function using the Adam optimizer in the next stage. The process of updating the trainable parameters using the Adam optimizer is described by the following equation, as proposed by [45].

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\alpha_{t+1} m_{t+1}}{\sqrt{v_{t+1}} + \epsilon}, \tag{18}$$

where α_t is the adaptive learning rate, with a default initial value of $\alpha_t = 0.001$, and is described as follows

$$\alpha_{t+1} = \alpha_t \frac{\sqrt{1-\beta_t^1}}{1-\beta_t^2}, \tag{19}$$

and

$$\begin{aligned} m_{t+1} &= \beta^1 m_t + (1-\beta^1)\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_t), \\ v_{t+1} &= \beta^2 v_t + (1-\beta^2)[\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_t)]^2. \end{aligned} \tag{20}$$

Both m_t and v_t are initialized to 0, and the other parameters are described as follows:

t: iteration (0, 1, 2, …, max iteration),

$\boldsymbol{\theta}_t$: trainable parameters (weights and biases) in the current iteration,

$\beta^1, \beta^2 \in [0, 1)$: exponential decay rates for moment estimates,

ϵ: very small number to prevent divided by zero,

$\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_t)$: gradient of the loss function regarding the trainable parameters.

The ANN architecture is iteratively improved by employing the Adam optimizer to generate a new $\boldsymbol{\theta}_{t+1}$. This process is repeated until a termination condition is met. The termination condition may be defined by reaching a specific number of iterations or by satisfying a predetermined error threshold between consecutive solutions. The complete process of r-PINN integrated with TSVD is summarized in Algorithm 1 and 2.

Algorithm 1 PINN-TSVD Algorithm for Solving PDEs

Require: the PDE system (e.g., Eq. (1));
Require: number of training points $N_{\mathcal{F}}$, N_0, N_b;
Require: training inputs x_i, y_j, t_k from each $N_{\mathcal{F}}$, N_0, and N_b;
Require: total number of layers m;
Require: maximum number of training iterations T;
Require: TSVD rank r and TSVD layer $= l_{tsvd}$;
Set $\mathbf{I} \leftarrow [x_i, y_j, t_k]^T$ as an input;
Initialize network parameters $\theta = \{\mathbf{W}, \mathbf{b}\}$;
for $t = 1$ to T **do**
    $\mathbf{h}_1 \leftarrow f(\mathbf{W}_1\mathbf{I} + \mathbf{b}_1)$;
    for $l = 2$ to m **do**
        if $l = l_{tsvd}$ **then**
            $\mathbf{W}_l \leftarrow \mathbf{U}_r^l \Sigma_r^l (\mathbf{V}_r^l)^T$;
            $\mathbf{h}_l \leftarrow f(\mathbf{W}_l\mathbf{h}_{l-1} + \mathbf{b}_l)$;
        else
            $\mathbf{h}_l \leftarrow f(\mathbf{W}_l\mathbf{h}_{l-1} + \mathbf{b}_l)$;
        end if
    end for
    Compute output layer:
    $\hat{u}(\mathbf{I}; \theta) \leftarrow \mathbf{W}_u\mathbf{h}_m$;
    Compute derivatives of $\hat{u}$ via automatic differentiation;
    Compute loss $\mathcal{L}(\theta_t)$;
    Update θ_t using Adam optimizer;
end for
$\theta^* \leftarrow$ parameters minimizing loss;
Return approximate outputs $\hat{u}$ over the domain;

Algorithm 2 Restarting Strategy of PINN-TSVD

Require: PINN-TSVD training procedure (Algorithm 1);
Require: Maximum number of cycles S;
Require: Maximum iterations per cycle T;
Require: Patience length N (no. of steps without improvement);
Set cycle index $s \leftarrow 1$;
Initialize weights and biases θ_0;
while $s \leq S$ **do**
    Initialize best loss $\mathcal{L}^b \leftarrow \infty$ and counter $n \leftarrow 0$;
    for $t = 1$ to T **do**
        Train using Algorithm 1 forward pass with current θ_t;
        if $\mathcal{L}(\theta_t) < \mathcal{L}^b$ **then**
            $\mathcal{L}^b \leftarrow \mathcal{L}(\theta_t)$;
            Save θ_t as best parameters θ^b;
            $n \leftarrow 0$;
        else
            $n \leftarrow n + 1$;
        end if
        if $n = N$ **then**
            Break the current iteration loop;
        end if
    end for
    Set $\theta_0 \leftarrow \theta^b$;
    $s \leftarrow s + 1$;
end while
Select final $\theta^* \leftarrow \theta^b$ (best from all cycles);
Return approximate solution using θ^*;

3.6 Another Performance Evaluation Metrics

Besides the MSE-based loss function, the mean absolute error (MAE) is also employed as a performance evaluation metric in this study. However, unlike the MSE, which measures the model's residuals with respect to the PDE (including initial and boundary residuals within the PINN formulation), MAE directly measures the average absolute difference between the predicted solution and the exact DE solution, providing a clear and interpretable metric of approximation accuracy. Specifically, the absolute error (AE) is first computed for each variable, representing the pointwise difference between the predicted and exact values, as formulated below:

$$AE_{=} \left| \hat{u}(x, y, t; \theta) - u_{\text{exact}}(x, y, t) \right|. \tag{21}$$

Here, $u_{\text{exact}}(x, y, t)$ represents the benchmark solution obtained from the analytical or ground-truth solution. Then, MAE is defined as the mean value of these AE over the domain, serving as a global measure of prediction accuracy. Calculating MAE can be formulated as:

$$MAE = \frac{1}{N} \sum_{i=1}^{N} \left(AE^{(i)} \right), \tag{22}$$

where N denotes the total number of points alongside the result domain. In this way, the AE provides localized error information, while the MAE summarizes the overall error magnitude across the domain.

4 Experimental Results

In this section, benchmark problems derived from up to three-dimensional PDEs and one system of PDEs are discussed. Table 1 provides a comprehensive list of all benchmark problems employed, along with references to scholarly articles that have previously addressed similar problems. The results of the analysis have been systematically documented in a series of tables and illustrative graphs. All computational experiments were conducted using the Python programming language and executed on a Windows 11 laptop. The laptop specifications include 8GB of RAM, an Intel Core i7-8565U processor, and an Intel UHD 620 graphics card. This hardware configuration ensured optimal performance during the execution of our experiments.

4.1 Simulation of TSVD Implementation within the ANN Architecture

To evaluate the impact of TSVD within the PINN architecture, two simulation experiments are conducted. The first examines how the position and number of TSVD layers affect accuracy and training speed, while the second explores how different rank-r values influence model performance. These simulations collectively offer a comprehensive understanding of the practical trade-offs introduced by TSVD in the PINN framework.

4.1.1 TSVD layer position simulation within the architecture

TSVD can be implemented at various positions within the hidden layers of the network. The placement and number of TSVD layers within the architecture may influence the approximation results, potentially affecting both accuracy and computational time. Therefore, before

Table 1 Benchmark PDE problems

Cases	Equation	Range	Initial and boundary conditions	Used in
One-dimensional PDE	$u_t - u_{xx} = 0$	$(x, t) \in [0, 1]$	$\begin{cases} u(x, 0) = sin(\pi x), \\ u(0, t) = u(1, t) = 0 \end{cases}$	[59]
Two-dimensional PDE	$u_{xx} + u_{yy} = r(x, y)$	$(x, y) \in [0, 1]$	$\begin{cases} u(x, 0) = xe^{-x}, \\ u(x, 1) = (x + 1)e^{-x}, \\ u(0, y) = y^3 \\ u(1, y) = (1 + y^3)e^{-1}, \end{cases}$	[60]
Three-dimensional PDE	$u_{xx} + \rho u_x - u_{yy} - u_{zz} = q(x, y, z, u)$	$(x, y, z) \in [0, 2]$	$\begin{cases} u(0, y, z) = (1 - cos(\pi y))(1 - cos(\pi z)), \\ u_x(0, x, y) = -\alpha(1 - cos(\pi y))(1 - cos(\pi z)), \\ u(x, 0, z) = u(x, 2, z) = u(x, y, 0) = u(x, y, 2) = 0 \end{cases}$	[61]
System of PDEs	$\begin{cases} \frac{\partial(uu)}{\partial x}(x, y) + \frac{\partial(uv)}{\partial y}(x, y) = \frac{1}{Re}\left(\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2}\right) - \frac{\partial p(x,y)}{\partial x} \\ \frac{\partial(uv)}{\partial x}(x, y) + \frac{\partial(vv)}{\partial y}(x, y) = \frac{1}{Re}\left(\frac{\partial^2 v(x,y)}{\partial x^2} + \frac{\partial^2 v(x,y)}{\partial y^2}\right) - \frac{\partial p(x,y)}{\partial y} \\ \frac{\partial u(x,y)}{\partial x} + \frac{\partial v(x,y)}{\partial y} = 0 \end{cases}$	$\begin{cases} x \in [0, 1] \\ y \in [-1, 0] \end{cases}$	$\begin{cases} u(0, y) = v(0, y) = 0 \\ u(1, y) = v(1, y) = 0 \\ u(x, -1) = v(x, -1) = 0 \\ u(x, 0) = 1, v(x, 0) = 0 \end{cases}$	[62]

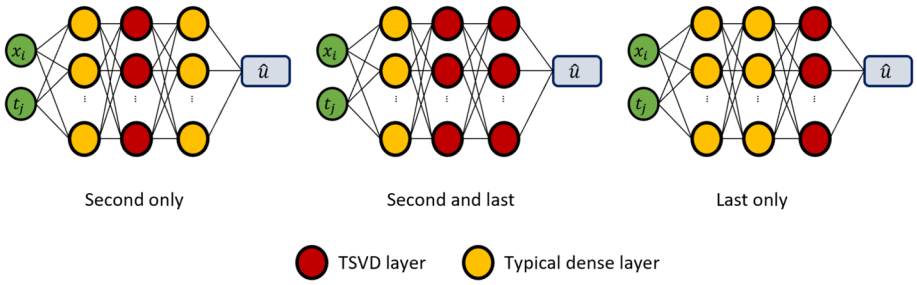


Fig. 4 Variety of TSVD layer simulation within the ANN architecture

analyzing the numerical results of the benchmark problems, preliminary simulations were conducted to determine the optimal number and placement of TSVD layers for effective integration within the PINN architecture. In this study, the test problem used for the simulation is the one-dimensional wave equation, as described in Table 1, due to its moderate complexity and its relevance as a baseline benchmark problem. The simulations explored various TSVD configurations, including placement in the second layer only, the last layer only, and both the second and last layers. An illustration of these configurations is shown in Figure 4.

It should be noted that the primary objective of TSVD is to compress the weight matrix to accelerate the training process. Therefore, TSVD is not applied to the first hidden layer in this simulation. This is because the number of trainable parameters between the input layer and the first hidden layer is already small, so applying TSVD at this layer would not provide any significant benefit in terms of model size reduction. In fact, it could add extra computation without meaningfully improving the model's efficiency. Such an approach would therefore contradict the objective of optimizing the networks size and accelerating training. Additionally, Table 2 outlines the configuration settings used in these simulations.

Table 2 illustrates that the ANN architecture comprises three hidden layers, each containing 50 neurons. This architecture was selected through a trial-and-error process to identify the optimal configuration. The input layer consists of two neurons corresponding to the spatial and temporal variables, x and t, respectively. The output layer contains a single neuron representing the approximate unknown function $\hat{u}(x, t)$. Furthermore, the total domain grid points for this problem are set to 200×200, resulting in a total of 40,000 points. Among these, only 2,500 points are selected as collocation training points, while 100 points are designated for initial and boundary training, with 25 points sampled from each initial and boundary condition.

Table 2 also shows that the maximum TSVD rank was temporarily set to 5. This value serves as a preliminary choice to evaluate the effect of the TSVD layer position, before a more detailed analysis of rank variation is conducted in the next simulation. A clear comparison of the number of trainable parameters between the TSVD-integrated method and the non-TSVD approaches (basic PINN and r-PINN) is presented in Table 3.

Table 3 shows that integrating TSVD with a truncated top-5 rank in one of the hidden layers significantly reduces the number of trainable parameters, from 2,550 to only 555. This reduction results from decomposing the standard fully connected weight matrix with rank 5, where the transformation matrix is expressed as:

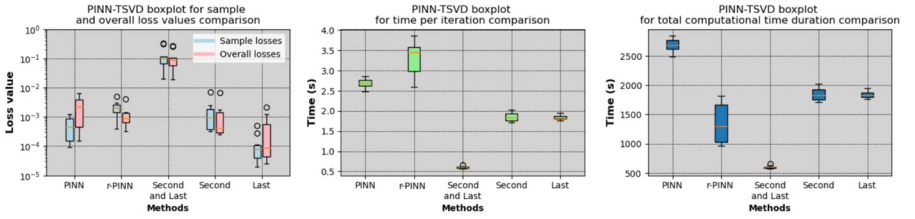
$$\mathbf{W}_l \approx \mathbf{W}_l^5 = \mathbf{U}_5 \boldsymbol{\Sigma}_5 \mathbf{V}_5^T,$$

Table 2 Configuration settings used in TSVD simulation training for one-dimensional wave equation

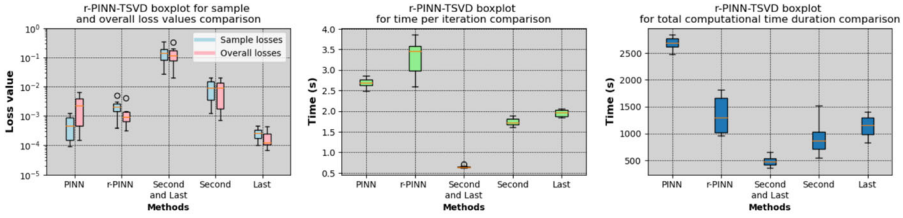
Method	ANN architecture	Total domain points	Col. training points	Init/boundary training points	TSVD rank	N patience	Cycle	Max iter	Initial learning rate
PINN-TSVD	2+(50 × 5)+1	200 × 200	2500	100	5	–	–	1000	1×10^{-3}
r-PINN-TSVD	2+(50 × 5)+1	200 × 200	2500	100	5	10	10	100	1×10^{-3}

Table 3 Comparison of trainable parameters between the network with a TSVD layer (rank = 5) and the standard (non-TSVD) network for one-dimensional wave equation

Layer type	Neurons	Number of trainable parameter			
		non-TSVD	Second only	Last only	Second and last
Input	2	0	0	0	0
Hidden layer 1	50	150	150	150	150
Hidden layer 2	50	2,550	**555**	2,550	**555**
Hidden layer 3	50	2,550	2,550	**555**	**555**
Output	1	51	51	51	51
Total parameters		**5301**	**3306**	**3306**	**1311**
Size		**99kB**	**78kB**	**78kB**	**61kB**



(a) basic PINN-TSVD.



(b) r-PINN-TSVD.

Fig. 5 Loss values (left), computational time per iteration (mid), and total computational time (right) comparisons between basic PINN, r-PINN, and all TSVD configurations in solving one-dimensional wave equation

where $\mathbf{U}_5 \in \mathbb{R}^{50\times5}$, $\mathbf{\Sigma}_5 \in \mathbb{R}^5$, and $\mathbf{V}_5^T \in \mathbb{R}^{5\times50}$. Including an additional bias vector with 50 elements leads to

$$(50 \times 5) + 5 + (5 \times 50) + 50 = 555$$

trainable parameters within the TSVD layer. In this implementation, the singular values are stored as a vector of length 5, since the off-diagonal elements of the singular value matrix are always zero, making it unnecessary to store the entire matrix explicitly. This substantial reduction also contributes to a smaller model memory size. The results of these simulation configurations, compared to the performance of the non-TSVD methods, are presented in Figure 5.

Each configuration shown in Figure 5 is tested through 10 independent simulation runs, which further validate the reliability and consistency of this approach in solving the problem. Among the TSVD configurations, the "last only" setup consistently yields lower loss values than the "second and last" or "second only" configurations, demonstrating its efficiency and

Table 4 Comparison of trainable parameters within the networks using TSVD with different rank values for solving one-dimensional wave equation

Layer type	Neurons	Number of trainable parameter			
		non-TSVD	rank = 5	rank = 10	rank = 15
Input	2	0	0	0	0
Hidden layer 1	50	150	150	150	150
Hidden layer 2	50	2550	2550	2550	2550
Hidden layer 3	50	2550	**555**	**1060**	**1565**
Output	1	51	51	51	51
Total parameters		**5,301**	**3,306**	**3,811**	**4,316**
Size		**99kB**	**78kB**	**85kB**	**91kB**

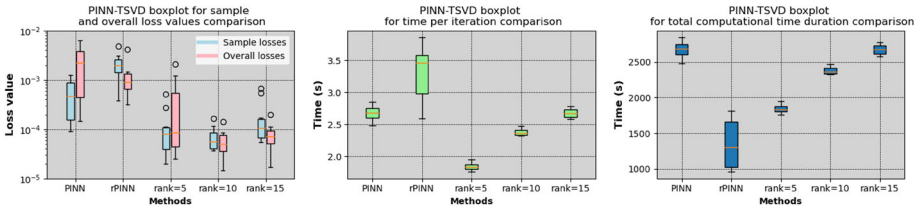
effectiveness in minimizing error. This trend remains consistent across both PINN-TSVD and r-PINN-TSVD cases, highlighting the robustness of applying TSVD to the last layer. Furthermore, when assessing computational efficiency, the middle and right plots demonstrate that incorporating TSVD effectively reduces computational costs. Although the "last only" configuration is not the fastest among the TSVD setups, it still achieves a notable reduction in computational time compared to the non-TSVD approaches, while maintaining lower loss. Given its ability to strike a balance between accuracy and computational efficiency, the "last only" configuration emerges as the most favorable choice for optimizing performance in this study.

4.1.2 TSVD rank simulation within the architecture

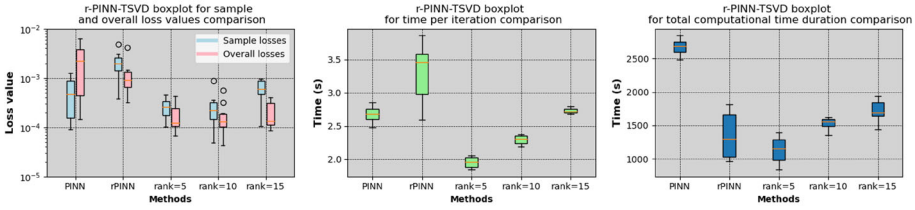
The next simulation evaluates the influence of different TSVD rank values within the architecture. Specifically, rank values of 5, 10, and 15 are selected to assess the trade-off between computational time and approximation accuracy. In this simulation, the PINN-TSVD model adopts the "last only" configuration, which was identified as the most practical choice in the previous experiment. This setup maintains a consistent network structure, allowing the effect of different TSVD ranks to be evaluated more clearly. Before presenting the simulation results, Table 4 provides a comparison of the number of trainable parameters associated with each rank setting.

Table 4 shows that the non-TSVD model has 5,351 trainable parameters. In contrast, applying the TSVD layer in the "last-only" configuration with ranks 5, 10, and 15 significantly reduces the parameter count to 3,356, 3,861, and 4,366, respectively. This demonstrates substantial compression without changing the overall architecture. To further assess its impact, Figure 6 compares iteration times, total training times, and final loss values over 10 independent runs.

As shown in Figure 6, increasing the TSVD rank value leads to a gradual rise in both time per iteration (mid) and total training time (right). This observation is consistent with the increase in the number of trainable parameters presented in Table 4. However, this additional computational cost is not accompanied by a significant improvement in model accuracy. As evidenced in Figure 6 (left), the loss values for TSVD models with ranks of 10 and 15 exhibit no significant improvement and tend to stagnate. Overall, the results from the two simulations evaluating TSVD layer position and rank configuration indicate that the 'last-



(a) basic PINN-TSVD.



(b) r-PINN-TSVD.

Fig. 6 Loss values (left), computational time per iteration (mid), and total computational time (right) comparisons between basic PINN, r-PINN, and all TSVD rank configurations in solving one-dimensional wave equation

only' configuration with a rank of 5 achieves the best balance between performance and efficiency. This configuration will therefore be adopted throughout this study.

4.2 One-dimensional Linear Wave Equation

For the one-dimensional PDE case, a PDE with one spatial variable (e.g., x) and possibly a temporal variable (t) is typically selected. The linear wave equation, a representative of the hyperbolic type, is employed to model the one-dimensional PDE. The wave equation is a fundamental model that appears in various scientific fields, such as acoustic wave propagation [63] and seismic wave propagation [64]. The analytical solution to this equation is provided in Equation 23 and visualized in Figure 7.

$$u(x,t) = \frac{1}{2}sin(\pi x)cos(\pi t) + \frac{1}{3}sin(3\pi x)sin(3\pi t). \tag{23}$$

Figure 7 also illustrates the randomly selected collocation, initial, and boundary training points. Specifically, 25 sample points are taken from each side of the boundary, resulting in 100 points in total. In addition, 2,500 collocation points are randomly sampled from a total of 200 $\times$ 200 grid points within the domain $(x,t) \in [0,1]$. The architecture of the ANN model for this problem was configured with three hidden layers, each containing 50 neurons, consistent with the previous simulations. A clear comparison of the number of trainable parameters between the TSVD-integrated and non-TSVD methods is shown in Table 5.

The approximate PINN solution for the one-dimensional wave equation is defined as follows:

$$\mathcal{F} := \hat{u}_{tt}(x,t;\boldsymbol{\theta}) - \hat{u}_{xx}(x,t;\boldsymbol{\theta}) = 0. \tag{24}$$

Furthermore, the loss function of this problem is defined by the following equation:

$$\mathcal{L}(\boldsymbol{\theta}) = MSE_0 + MSE_b + MSE_{\mathcal{F}}, \tag{25}$$

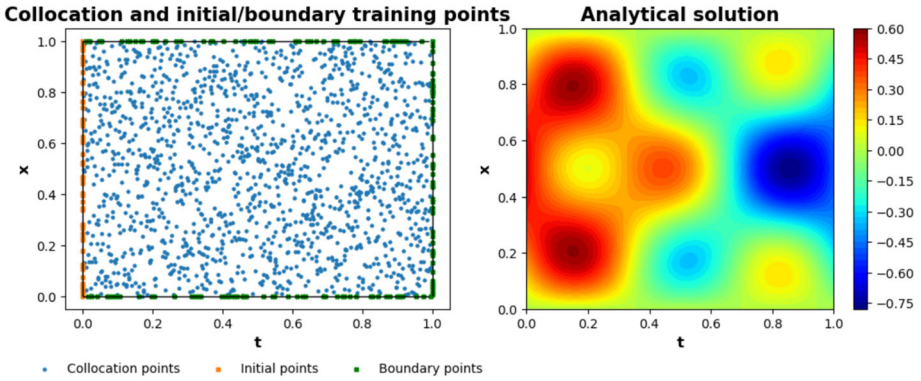


Fig. 7 Randomly selected collocation, initial, and boundary training points within the overall domain grid (left) and analytical solution (right) for one-dimensional wave equation

Table 5 Trainable parameters comparison between method with TSVD and non-TSVD in one-dimensional wave equation

Layer type	Neurons	Number of trainable parameter	
		non-TSVD	With TSVD with $rank = 5$
Input	2	0	0
Hidden layer 1	50	150	150
Hidden layer 2	50	2,550	2,550
Hidden layer 3	50	2,550	555
Output	1	51	51
Total parameters		**5301**	**3306**
Size		**99kB**	**78kB**

where

$$\begin{cases} MSE_0 = \frac{1}{N_0}\sum_{i=1}^{N_0}\left[\hat{u}(x_0^i, 0; \boldsymbol{\theta}) - \frac{1}{2}sin(\pi x_0^i)\right]^2 + \left[\hat{u}_t(x_0^i, 0; \boldsymbol{\theta}) - \pi sin(3\pi x_0^i)\right]^2, \\ MSE_b = \frac{1}{N_b}\sum_{i=1}^{N_b}\left[\hat{u}(0, t_b^i; \boldsymbol{\theta})\right]^2 + \left[\hat{u}(1, t_b^i; \boldsymbol{\theta})\right]^2, \\ MSE_{\mathcal{F}} = \frac{1}{N_{\mathcal{F}}}\sum_{i=1}^{N_{\mathcal{F}}}\left[\mathcal{F}(x_{\mathcal{F}}^i, t_{\mathcal{F}}^i; \boldsymbol{\theta})\right]^2. \end{cases} \tag{26}$$

This loss function should be minimized during the training process to obtain the best approximate solution. However, the primary concern in the construction of a PINN model is the randomness of weight initialization, which can frequently result in poor convergence if the initial weights are not appropriately chosen [65]. Running the models multiple times is crucial to emphasize their stability in terms of convergence and effectiveness [66]. Thus, 10 independent runs are performed to ensure the stability and consistency in learning capability of each method. For the restarting approaches, 100 iterations are conducted across 10 separate cycles. In comparison, the non-restarting approaches is subjected to 1,000 iterations in a single sequence. The visualization summary of these multiple runs can be seen in Figure 8.

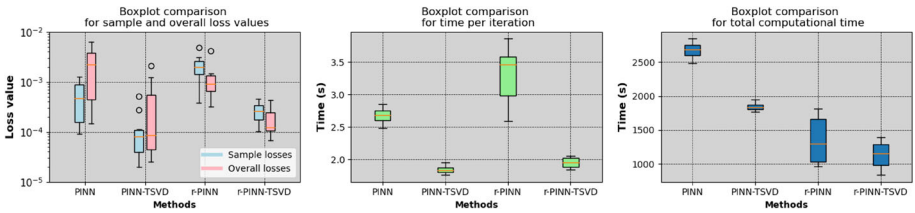


Fig. 8 Boxplot comparison of loss values, time per iteration, total computational time from 10 independent runs between the basic PINN, PINN-TSVD, r-PINN, and r-PINN-TSVD in solving one-dimensional wave equation

Table 6 Minimum, maximum, mean, and standard deviation for all methods during ten times run in one-dimensional wave equation

Stat	Basic PINN		Basic PINN-TSVD		r-PINN		r-PINN-TSVD	
	Sample	Overall	Sample	Overall	Sample	Overall	Sample	Overall
Min	9.101e−05	1.482e−04	**1.955e−05**	**2.470e−05**	3.859e−04	3.228e−04	**1.009e−04**	**6.808e−05**
Max	1.251e−03	6.311e−03	**5.084e−04**	**2.094e−03**	4.866e−03	4.153e−03	**4.489e−04**	**4.226e−04**
Mean	5.407e−04	2.547e−03	**1.287e−04**	**4.660e−04**	2.102e−03	1.231e−03	**2.625e−04**	**1.812e−04**
Std. Dev.	4.425e−04	2.341e−03	**1.518e−04**	**6.893e−04**	1.271e−03	1.090e−03	**1.140e−04**	**1.142e−04**

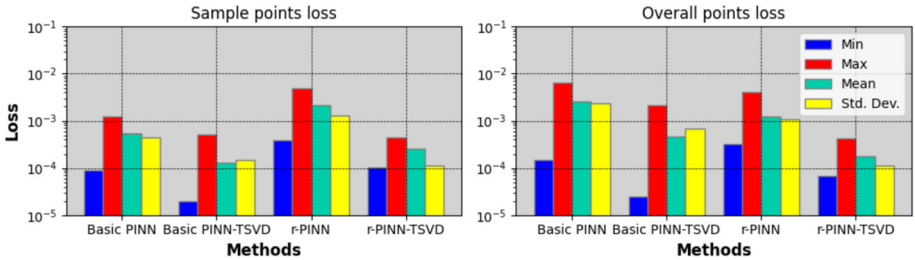


Fig. 9 Visualization of the statistical description of loss function values for all methods over ten runs on the one-dimensional wave equation

During the 10 independent runs, the boxplot above shows that integrating TSVD consistently yields lower loss function values compared to the non-TSVD methods. This is primarily due to the lower sample loss and overall loss values obtained by both TSVD methods compared to the non-TSVD method, as illustrated in Figure 8 (left). This statement is further supported by Table 6, which presents statistical details across all iterations for each method, with bold values indicating better results.

From Table 6, it is observed that the minimum, maximum, and average (mean) values associated with the TSVD-integrated method are lower than those of the non-TSVD method. Moreover, the standard deviation values are also lower in the TSVD-integrated method, indicating more stable performance across all runs for the one-dimensional wave equation case. Furthermore, Figure 9 visualizes the values in Table 6 to provide a clearer illustration.

Despite a reduction in loss values, a substantial decrease in computational time was also observed when incorporating TSVD within the PINN framework. This reduction is clearly attributed to the significant decrease in the total number of trainable parameters, as shown in Table 5. A clearer illustration of the computational time statistics is presented in Figure 10.

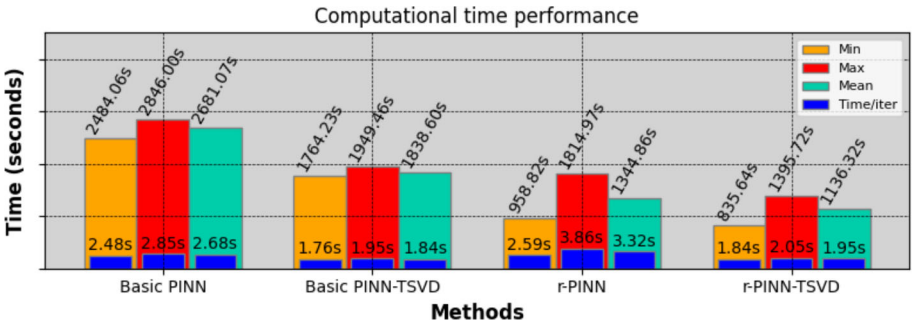
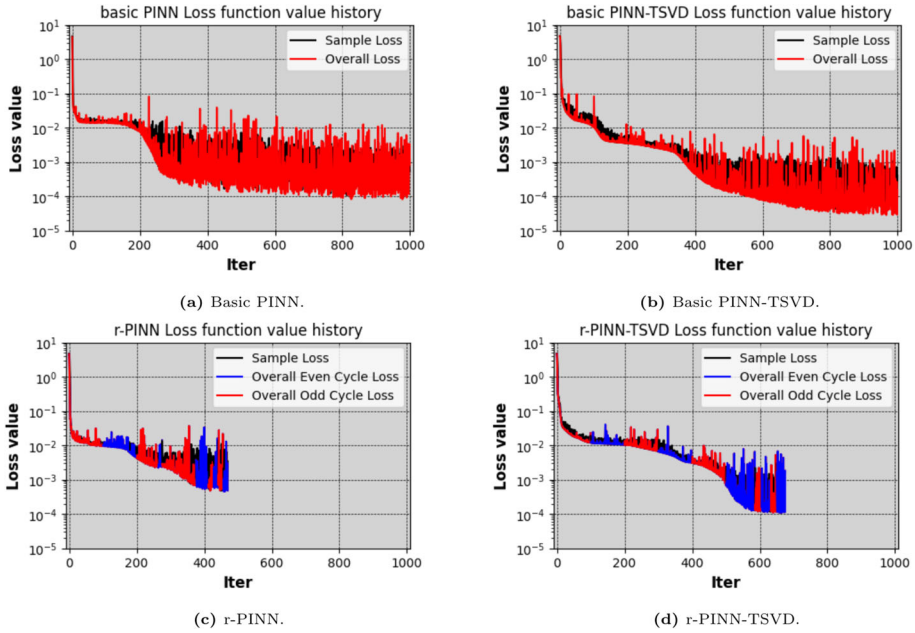


Fig. 10 Comparison of computational performance among all methods for the one-dimensional wave equation



(a) Basic PINN. (b) Basic PINN-TSVD.

(c) r-PINN. (d) r-PINN-TSVD.

Fig. 11 Loss function value history for each method in one-dimensional wave equation

When applied to the basic PINN, the integration of TSVD markedly reduces the average time per iteration from 2.68 seconds to 1.84 seconds, representing a 31.34% reduction. Specifically, it reduces the total computational duration from 2861.07 seconds in the basic PINN to 1838.60 seconds in the PINN-TSVD variant, amounting to a substantial 35.73% reduction. On the other hand, TSVD also brings a significant improvement in the r-PINN, reducing the time required per iteration from 3.32 seconds to 1.95 seconds, representing a 41.2%. This further demonstrates the computational efficiency of incorporating TSVD in solving the one-dimensional wave equation. Furthermore, Figure 11 offers a comprehensive overview of the progression of the loss function captured at each iteration for all methods.

Notably, in the absence of a restarting approach (as shown in Figures 11a and 11b), the training process continues up to the 1000th iteration even when no significant reduction occurs in the loss function value. This may lead to instability in subsequent iterations and

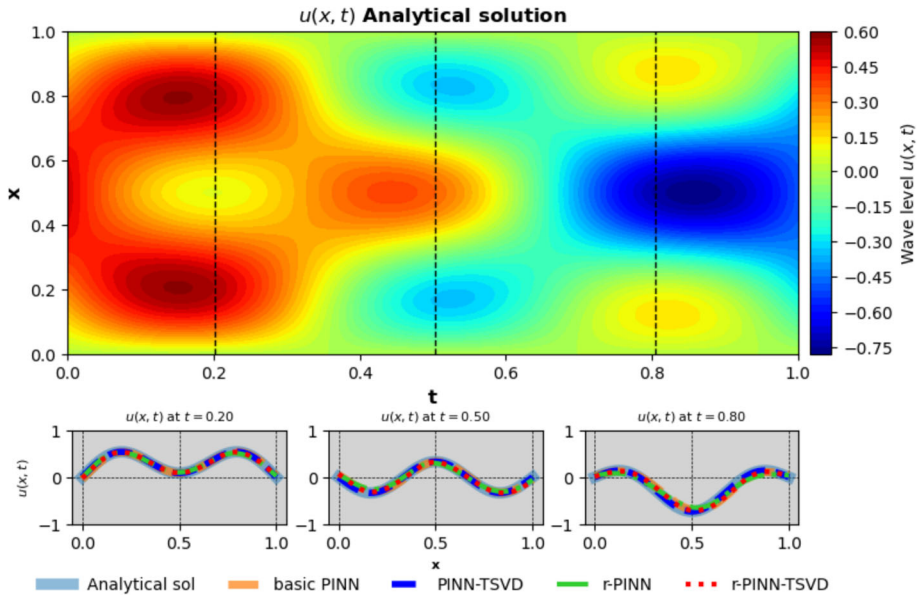


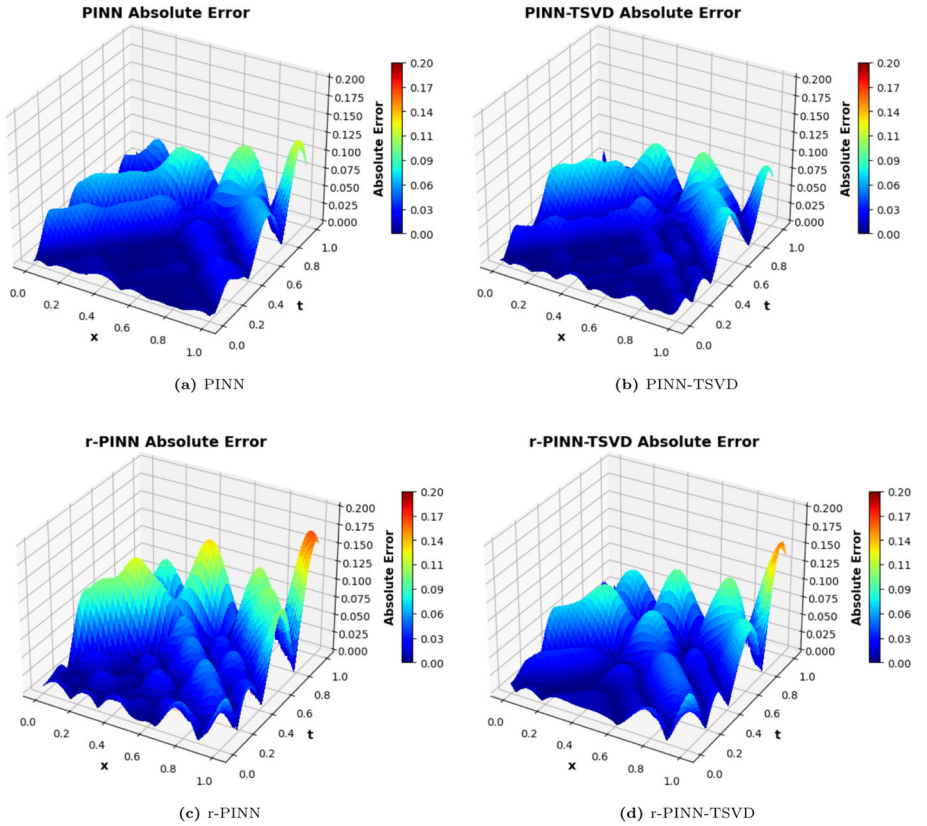
Fig. 12 Analytical solution and comparison between the basic PINN, PINN-TSVD, r-PINN, and r-PINN-TSVD solutions for the one-dimensional wave equation at $t = 0.20$, $t = 0.50$ and $t = 0.80$ (bottom)

result in a higher final loss. Conversely, by employing a restarting strategy, as reflected in the even and odd training cycles, the training process is halted once no significant progress is observed in the loss function. This is due to the continuous monitoring of the loss function at each iteration. Upon completion of the training process, the approximation results obtained for all methods are displayed in Figure 12.

The top panel of Figure 12 displays the spatiotemporal solution from the analytical model, while the bottom subfigures provide cross-sectional comparisons at selected time instances: $t = 0.20$, $t = 0.50$, and $t = 0.80$. At each time slice, the predicted results from both TSVD and non-TSVD methods closely align with the analytical solution, capturing the essential dynamics of the PDE. Given the close visual agreement among these approximations, a more detailed comparison is presented in Figure 13, which quantifies the absolute differences between all tested methods relative to the reference solution.

A uniform error scale is applied to all AE plots to enable a fair visual comparison among them. However, the distribution plots in Figure 13 alone are not sufficient to determine the overall differences among the methods. Therefore, a comparison of MAE, provided in Figure 14, offers a clearer illustration of each method's overall accuracy relative to the analytical solution.

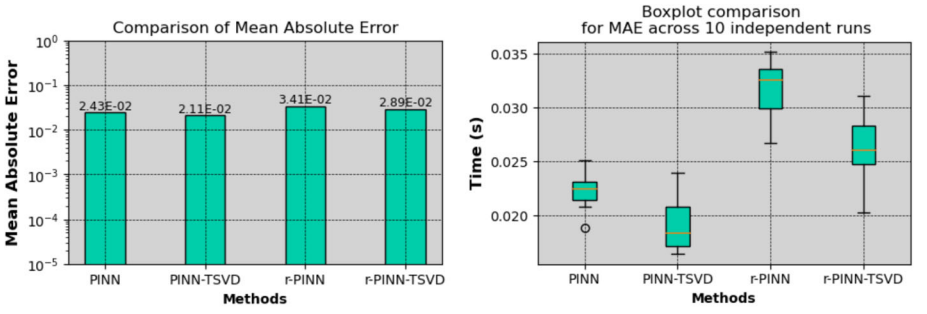
Figure 14a reaffirms the observations from the AE contour plots for each method. Incorporating TSVD within the PINN framework improves accuracy relative to the analytical solution, compared to the non-TSVD method. From the figure, PINN-TSVD achieves the lowest MAE of 2.11×10^{-2}, which is lower than its non-TSVD counterpart, which recorded 2.43×10^{-2}. An improvement is also observed when TSVD is applied to r-PINN, which achieves an MAE of 2.89×10^{-2}, improved from 3.41×10^{-2} obtained by the original r-PINN. This result is consistent across 10 independent runs, as shown in Figure 14b , where incorporating TSVD consistently improves accuracy compared to non-TSVD approaches.



(a) PINN (b) PINN-TSVD

(c) r-PINN (d) r-PINN-TSVD

Fig. 13 Comparison of the AE between (**a**) PINN, (**b**) PINN-TSVD, (**c**) r-PINN, and (**d**) r-PINN-TSVD relative to the analytical solution along the one-dimensional wave equation domain



(a) MAE (b) MAE across 10 independent runs

Fig. 14 Comparison of the MAE between the basic PINN, PINN-TSVD, r-PINN, and r-PINN-TSVD relative to the one-dimensional wave equation's analytical solution

Finally, Table 7 summarizes all previously presented results, further underscoring the efficacy of the TSVD approach in this one-dimensional wave equation case.

4.3 Two-dimensional Linear Poisson Equation

The next evaluation uses the linear Poisson equation to assess all methods' performance in the two-dimensional case. The Poisson equation was chosen not only for representing a spatially two-dimensional problem, but also for being one of the most widely used PDEs in modern technologies [67, 68], and is applied to a wide range of problems, such as fluid flow simulation [69], surface reconstruction [70], and image processing [71]. The analytical solution for this equation is given in Equation 23.

$$u(x, y) = (x + y^3)e^{-x}. \tag{27}$$

A total of 40,000 grid points within the Poisson domain were established, consistent with the preceding one-dimensional scenario and illustrated in Figure 15.

From Figure 15, 100 random points were selected exactly on the boundary, along with 2,500 sample collocation points. Moreover, the input pairs x and y of the sample points are computed using an architecture configured with three hidden layers, each containing 50 neurons. Similar to the previous case, the TSVD layer was embedded within the third hidden layer. A comparison of the number of trainable parameters among all methods is consistent, as shown in Table 5 in the previous one-dimensional example. Additionally, the approximate solution to this PDE using the ANN is defined in Equation 28.

$$\mathcal{F} := \hat{u}_{xx}(x, y; \boldsymbol{\theta}) + \hat{u}_{yy}(x, y; \boldsymbol{\theta}) - r(x, y), \tag{28}$$

where

$$r(x, y) = e^{-x}(x - 2 + y^3 + 6y). \tag{29}$$

The loss function is defined in Equation (30) as follows

$$\mathcal{L}(\boldsymbol{\theta}) = MSE_b + MSE_{\mathcal{F}}, \tag{30}$$

where MSE_b and $MSE_{\mathcal{F}}$ can be replaced by the following expressions:

$$\begin{cases} MSE_b = \dfrac{1}{N_b}\displaystyle\sum_{i=1}^{N_b}\left[\hat{u}(x_b^i, 0; \boldsymbol{\theta}) - x_b^i e^{x_b^i}\right]^2 + \left[\hat{u}(x_b^i, 1; \boldsymbol{\theta}) - (x_b^i + 1)e^{-x_b^i}\right]^2 \\ \qquad\qquad + \left[\hat{u}(0, y_b^i; \boldsymbol{\theta}) - (y_b^i)^3\right]^2 + \left[\hat{u}(1, y_b^i; \boldsymbol{\theta}) - (1 + [y_b^i]^3)e^{-1}\right]^2, \\ MSE_{\mathcal{F}} = \dfrac{1}{N_{\mathcal{F}}}\displaystyle\sum_{i=1}^{N_{\mathcal{F}}}\left[\mathcal{F}(x_{\mathcal{F}}^i, y_{\mathcal{F}}^i; \boldsymbol{\theta})\right]^2. \end{cases} \tag{31}$$

The training process to minimize the loss function for approximating the two-dimensional PDE using r-PINN was conducted for 100 iterations, repeated across 5 cycles. Meanwhile, the basic PINN method was only performed for 500 iterations in a single sequence. The performance of each method across 10 independent training runs is visualized in Figure 16.

The Poisson equation is a fundamental equation that can be solved using PINN. Consequently, all four methods, whether with or without TSVD, are capable of achieving good and stable loss function performance across ten runs. However, upon further analysis, the integration of TSVD into both the basic PINN and r-PINN effectively minimizes overfitting and underfitting. This is indicated by the minimal gap between the sample loss and the overall

Table 7 One-dimensional wave equation summarize table

Method	Cycle	Stat	Total iter	Time/iter	Total time	Loss Sample	Overall	MAE
Basic PINN	1	Min	1000	2.48s	2484.06s	9.101$e-$05	1.482$e-$04	1.884$e-$02
		Max	1000	2.85s	2846.00s	1.251$e-$03	6.311$e-$03	2.513$e-$02
		Mean	1000	2.68s	2681.08s	5.407$e-$04	2.547$e-$03	2.235$e-$02
		Std. Dev.	0	0.11s	113.14s	4.425$e-$04	2.341$e-$03	**1.708e−03**
Basic PINN-TSVD	1	Min	1000	**1.76s**	**1764.23s**	**1.955e−05**	**2.470e-05**	**1.638e−02**
		Max	1000	**1.95s**	**1949.46s**	**5.084e−04**	**2.094e−03**	**2.387e−02**
		Mean	1000	**1.84s**	**1838.60s**	**1.288e−04**	**4.660e−04**	**1.920e−02**
		Std. Dev.	0	**0.056s**	**55.27s**	**1.518e−04**	**6.893e−04**	2.559$e-$03
r-PINN	10	Min	**313**	2.59s	958.82s	3.859$e-$04	3.228$e-$04	2.670$e-$02
		Max	**520**	3.86s	1814.97s	4.866$e-$03	4.153$e-$03	3.520$e-$02
		Mean	**411.2**	3.32s	1344.86s	2.102$e-$03	1.231$e-$03	3.186$e-$02
		Std. Dev.	**70.93**	0.44s	342.75s	1.272$e-$03	1.090$e-$03	**2.520e−03**
r-PINN-TSVD	10	Min	446	**1.84s**	**835.64s**	**1.009e−04**	**6.808e−05**	**2.026e−02**
		Max	755	**2.05s**	**1395.72s**	**4.489e−04**	**4.226e−04**	**3.114e−02**
		Mean	583	**1.95s**	**1136.32s**	**2.626e−04**	**1.812e−04**	**2.604e−02**
		Std. Dev.	100.55	**0.08s**	**206.47s**	**1.140e−04**	**1.143e−04**	3.087$e-$03

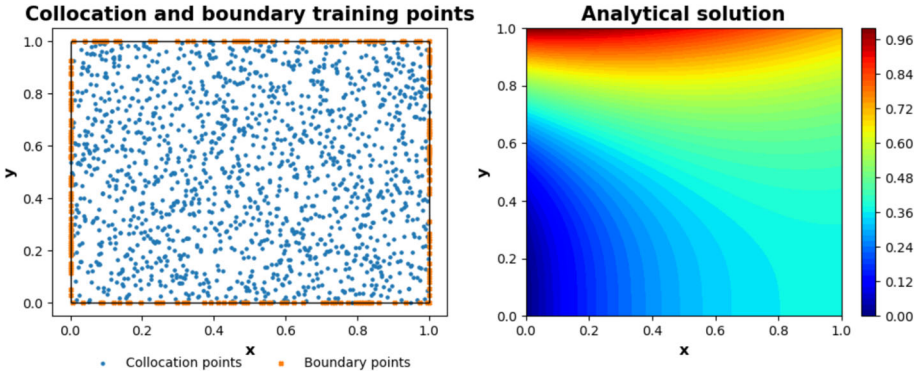


Fig. 15 Randomly selected collocation, initial, and boundary training points within the overall domain grid (left) and analytical solution (right) for the two-dimensional wave equation

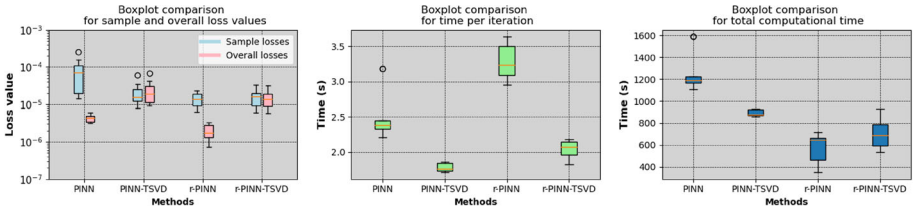


Fig. 16 Boxplot comparison of loss values, time per iteration, total computational time from 10 independent runs between the basic PINN, PINN-TSVD, r-PINN, and r-PINN-TSVD in solving two-dimensional Poisson equation

Table 8 Minimum, maximum, mean, and standard deviation for all methods during ten times run in two-dimensional Poisson equation

Stat	Basic PINN		Basic PINN-TSVD		r-PINN		r-PINN-TSVD	
	Sample	Overall	Sample	Overall	Sample	Overall	Sample	Overall
Min	$1.444e{-}05$	**3.167e-06**	**8.018e-06**	$9.263e{-}06$	$6.198e{-}06$	**7.365e-07**	**6.114e-06**	$5.870e{-}06$
Max	$2.601e{-}04$	**5.996e-06**	**6.005e-05**	$6.814e{-}05$	**2.355e-05**	**3.324e-06**	$3.340e{-}05$	$3.222e{-}05$
Mean	$8.502e{-}05$	**4.311e-06**	**2.184e-05**	$2.470e{-}05$	**1.432e-05**	**1.961e-06**	$1.603e{-}05$	$1.473e{-}05$
Std. Dev.	$7.919e{-}05$	$9.296e{-}07$	$1.587e{-}05$	$1.864e{-}05$	**6.405e-06**	**9.207e-07**	$8.214e{-}06$	$7.859e{-}06$

loss. In contrast, the non-TSVD methods exhibit a clearer gap in both the basic PINN and r-PINN approaches, as depicted in Figure 16 (left). The statistical details from ten iterations, including the minimum, maximum, mean, and standard deviation, are presented in Table 8. While TSVD-based methods show consistently lower loss values at the sample level, this advantage does not extend to the overall loss. The results in Table 8 are visualized in Figure 17.

Although methods incorporating TSVD do not maintain their performance in terms of overall loss for this case, they offer another advantage, as illustrated in Figure 18, which presents the computational time for all methods.

The computational time performance of all four methods was evaluated under conditions similar to the previous case. As shown in Figure 18, integrating TSVD consistently acceler-

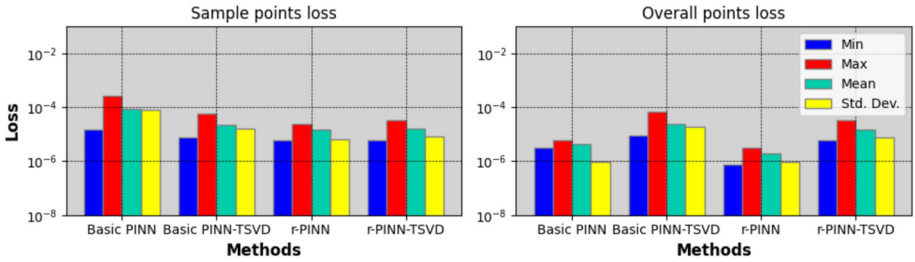


Fig. 17 Visualization of the statistical description of loss function values for all methods over ten runs on the two-dimensional Poisson equation

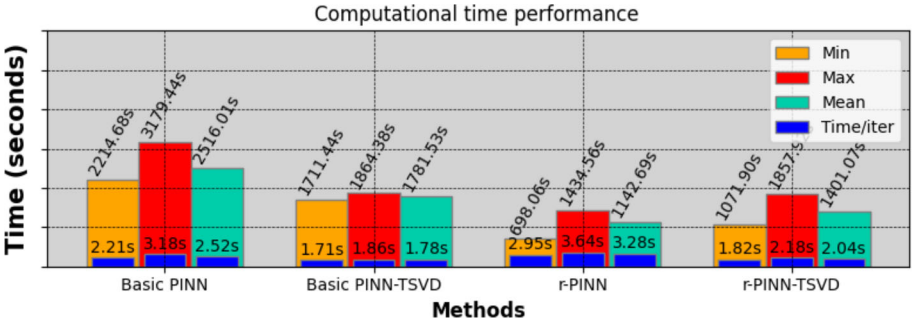


Fig. 18 Comparison of computational performance among all methods in two-dimensional Poisson equation

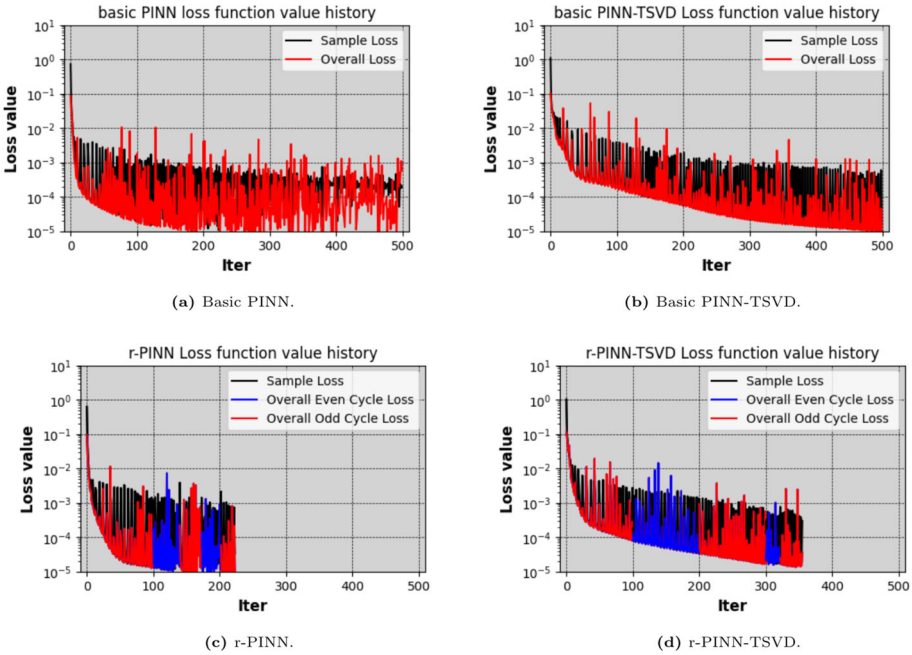
ates the time per iteration compared to non-TSVD methods. The average time per iteration for basic PINN is 2.52 seconds, which improves to 1.78 seconds, representing a 29.36% reduction. Similarly, r-PINN with TSVD reduces the time from 3.28 to 2.04 seconds, resulting in a 37.8% improvement. Although TSVD methods may not always yield the best overall loss, their efficiency in reducing computational time demonstrates a clear advantage in terms of training speed. The corresponding loss function histories are shown in Figure 19.

Figure 19 shows the loss function history for maximum 500 iterations for all four methods. r-PINN and r-PINN-TSVD converge faster than the basic methods, benefiting from the restarting strategy. TSVD-based methods, particularly PINN-TSVD and r-PINN-TSVD, exhibit more stable loss curves with fewer fluctuations. While they do not achieve the absolute lowest loss, their performance remains comparable. The corresponding approximation results for the 2D Poisson equation are shown in Figure 20.

As a fundamental PDE, the Poisson equation is closely approximated by all methods, resulting in indistinguishable differences among them, as observed in Figure 20. Therefore, Figure 21 compares their AE alongside the problem domain to highlight the absolute differences with the analytical solution.

Overall, Figure 21 shows that all methods tend to produce relatively random error spikes across the domain. It is difficult to determine which method shows closer agreement with the analytical solution based on this figure. Figure 22 presents the MAE comparison across all methods to better illustrate the accuracy of each method relative to the analytical solution.

As shown in Figure 22a , all methods achieve MAE values on the order of 10^{-4}, indicating that each method is capable of producing accurate approximations. On the other hand, Figure 22b illustrates the distribution of MAE across 10 independent runs. While there are slight



(a) Basic PINN. (b) Basic PINN-TSVD.

(c) r-PINN. (d) r-PINN-TSVD.

Fig. 19 Loss function value history for each method in two-dimensional Poisson equation

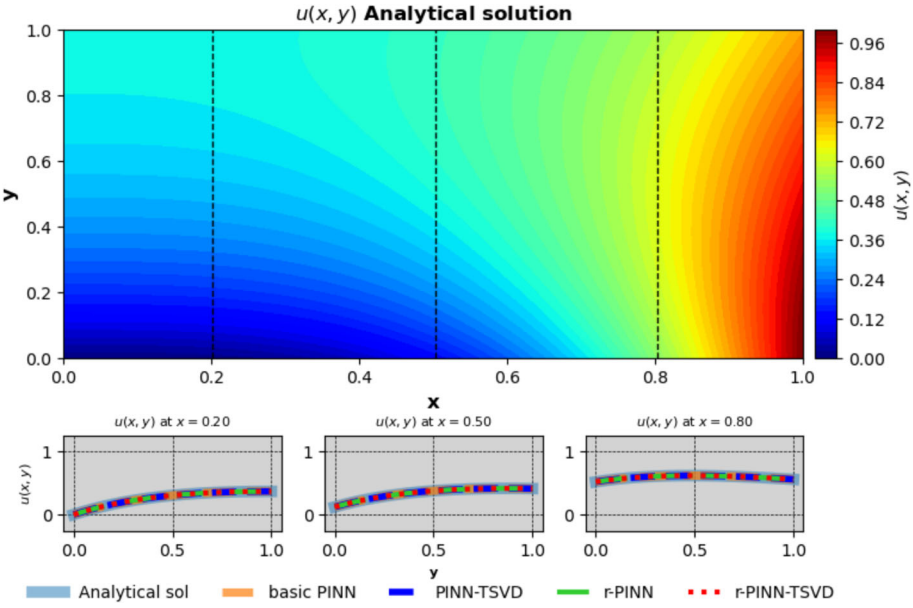
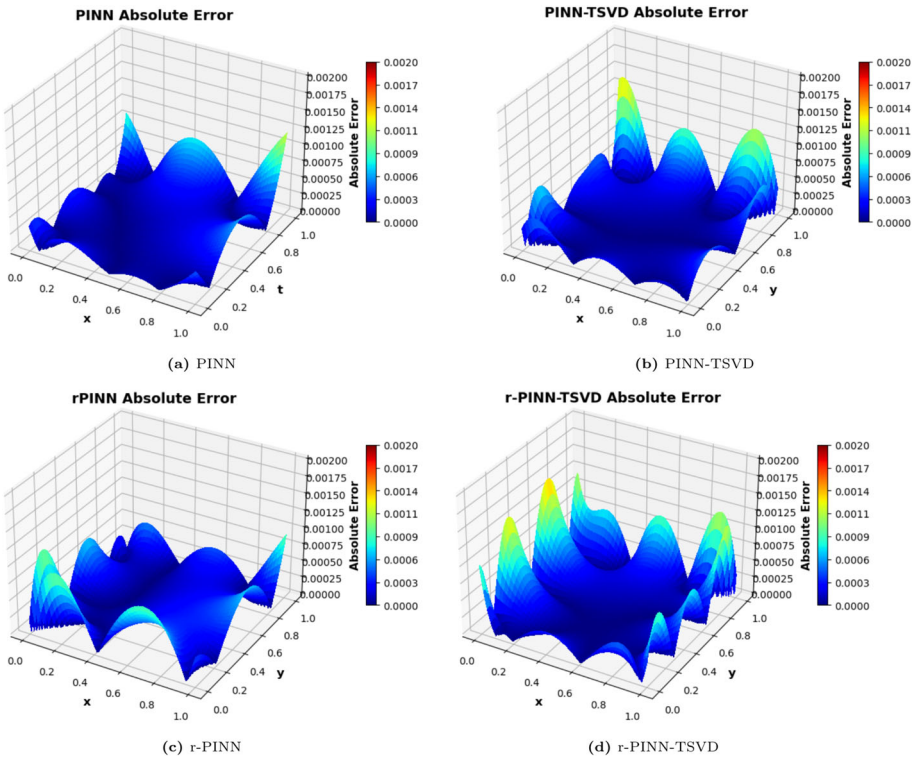


Fig. 20 Analytical solution and comparison between the basic PINN, PINN-TSVD, r-PINN, and r-PINN-TSVD solutions for the two-dimensional wave equation at $x = 0.20$, $x = 0.50$ and $x = 0.80$ (bottom)

Table 9 Two-dimensional Poisson equation summarize table

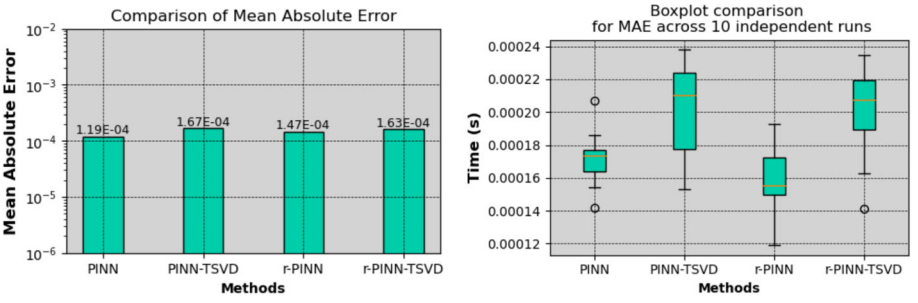
Method	Cycle	Stat	Total iter	Time/iter	Total time	Loss		MAE
						Sample	Overall	
Basic PINN	1	Min	500	2.21s	1107.34s	1.444$e-$05	**3.167e−06**	**1.417e−04**
		Max	500	3.18s	1589.72s	2.601$e-$04	**5.996e−06**	**2.068e−04**
		Mean	500	2.52s	1258.01s	8.502$e-$05	**4.311e−06**	**1.717e−04**
		Std. Dev.	0	0.36s	178.84s	7.919$e-$05	**9.296e−07**	**1.680e−05**
Basic PINN-TSVD	1	Min	500	**1.71s**	**855.72s**	**8.018e−06**	9.263$e-$06	1.530$e-$04
		Max	500	**1.86s**	**932.19s**	**6.005e−05**	6.814$e-$05	2.384$e-$04
		Mean	500	**1.78s**	**890.77s**	**2.184e−05**	2.470$e-$05	2.013$e-$04
		Std. Dev.	0	**0.06s**	**30.21s**	**1.587e−05**	1.864$e-$05	2.711$e-$05
r-PINN	5	Min	**96**	2.95s	**349.03s**	6.198$e-$06	**7.365e-07**	**1.186e−04**
		Max	**220**	3.64s	**717.28s**	**2.355e-05**	**3.324e−06**	**1.929e−04**
		Mean	**177.40**	3.28s	**571.34s**	**1.432e-05**	**1.961e−06**	**1.577e−04**
		Std. Dev.	**51.64**	0.26s	139.73s	**6.405e−06**	**9.207e-07**	**2.099e−05**
r-PINN-TSVD	5	Min	275	**1.82s**	535.95s	**6.114e−06**	5.870$e-$06	1.411$e-$04
		Max	446	**2.18s**	928.96s	3.340$e-$05	3.222$e-$05	2.345$e-$04
		Mean	342.90	**2.04s**	700.54s	1.603$e-$05	1.473$e-$05	2.003$e-$04
		Std. Dev.	61.62	**0.12s**	**134.78s**	8.214$e-$06	7.859$e-$06	2.815$e-$05



(a) PINN (b) PINN-TSVD

(c) r-PINN (d) r-PINN-TSVD

Fig. 21 Comparison of the AE between (**a**) PINN, (**b**) PINN-TSVD, (**c**) r-PINN, and (**d**) r-PINN-TSVD relative to the analytical solution along the two-dimensional Poisson equation domain



(a) MAE (b) MAE across 10 independent runs

Fig. 22 Comparison of the MAE between the basic PINN, PINN-TSVD, r-PINN, and r-PINN-TSVD relative to the two-dimensional Poisson equation's analytical solution

variations in accuracy, all methods remain within a close range, reflecting their comparable performance. The summary of all previous results is presented in Table 9.

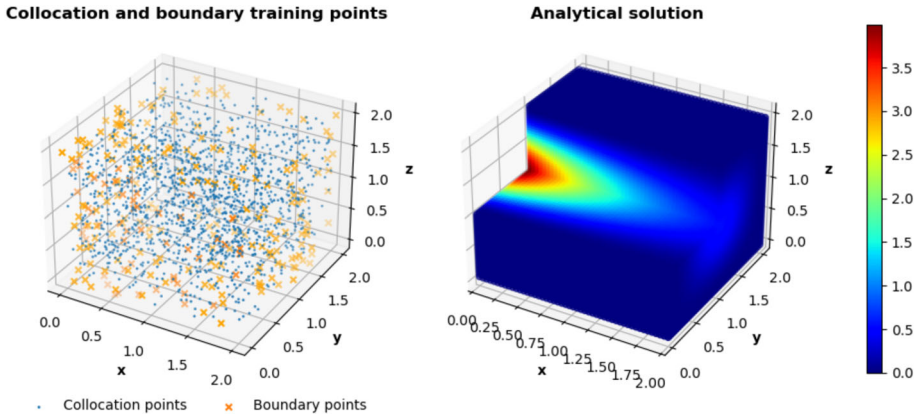


Fig. 23 Randomly selected collocation and boundary training points within the overall domain grid (left) and analytical solution (right) for the three-dimensional SG equation

4.4 Three-dimensional Sine-Gordon equation

In the three-dimensional case, the PDE involves three spatial dimensions (e.g., x, y, and z). The nonlinear hyperbolic sine-Gordon (SG) equation is utilized. The term "sine-Gordon" (SG) is a linguistic twist on the similarly named Klein–Gordon equation [72]. This equation is used in a variety of fields, including, but not limited to, magnetic flux propagation in Josephson junctions, sound propagation in crystal lattices, and several other areas [73, 74]. The analytical solution for this equation is given in Equation 32 [61].

$$u(x, y, z) = e^{-\alpha x}(1 - cos(\pi y))(1 - cos(\pi z)) \tag{32}$$

The total number of grid points within the SG domain is 125,000, derived from a $50 \times 50 \times 50$ grid over the x, y, and z dimensions. For this problem, 300 boundary points (50 from each side) and 2,500 collocation points are used. Figure 23 illustrates the grid-point layout alongside the analytical solution.

The configuration of the ANN architecture in the third case needs to be increased to 3 hidden layers with 200 neurons each, and the TSVD layer is embedded within the third hidden layer. The comparison of the number of trainable parameters among all three methods is shown in Table 10.

As the network expands, the number of trainable parameters correspondingly increases. The implementation of TSVD, where $rank = 5$, results in only 43,406 trainable parameters, compared to the usual 81,401. This also reduces the model's total memory requirement to just 169 kB, compared to the standard size of 317 kB without TSVD. Consequently, the approximate solution for this PDE is defined as follows:

$$\mathcal{F} := \hat{u}_{xx}(x, y, z; \boldsymbol{\theta}) + \rho\hat{u}_t(x, y, z; \boldsymbol{\theta}) - \hat{u}_{yy}(x, y, z; \boldsymbol{\theta}) - \hat{u}_{zz}(x, y, z; \boldsymbol{\theta}) - q(x, y, z, u), \tag{33}$$

Table 10 Trainable parameters comparison between method with TSVD and non-TSVD in three-dimensional SG equation

Layer type	Neurons	Number of trainable parameter	
		With TSVD with $rank = 5$	Without TSVD/basic PINN
Input	3	0	0
Hidden layer 1	200	800	800
Hidden layer 2	200	40200	40200
Hidden layer 3	200	2205	40200
Output	1	201	201
Total parameters		**43406**	**81401**
Size		**169kB**	**317kB**

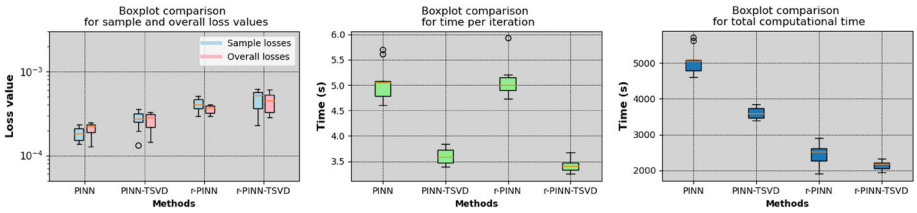


Fig. 24 Boxplot comparison of loss values, time per iteration, total computational time from 10 independent runs between the basic PINN, PINN-TSVD, r-PINN, and r-PINN-TSVD in solving the three-dimensional SG equation

where

$$\begin{aligned} q(x, y, z, u) = &-2\sin(u(x, y, z)) + 2\sin\left[e^{-\alpha x}(1-\cos(\pi y))(1-\cos(\pi z))\right] \\ &- e^{-\alpha x}\Big[\alpha(\rho-\alpha)(1-\cos(\pi y))(1-\cos(\pi z)) \\ &+ \pi^2\big(\cos(\pi y)+\cos(\pi z)-2\cos(\pi y)\cos(\pi z)\big)\Big] \end{aligned} \tag{34}$$

In this case, the values of α and ρ are set to 1. Moreover, the loss function is defined in Equation (35) as follows:

$$\mathcal{L}(\boldsymbol{\theta}) = MSE_b + MSE_{\mathcal{F}}, \tag{35}$$

where the MSE_b and $MSE_{\mathcal{F}}$ can be replaced by

$$\begin{cases} MSE_b = \dfrac{1}{N_b}\displaystyle\sum_{i=1}^{N_b}\Big[\hat{u}(0, y_b^i, z_b^i; \boldsymbol{\theta}) - (1-cos(\pi y_b^i))(1-cos(\pi z_b^i))\Big]^2 \\ \qquad + \Big[\hat{u}_x(0, y_b^i, z_b^i; \boldsymbol{\theta}) + \alpha(1-cos(\pi y_b^i))(1-cos(\pi z_b^i))\Big]^2 + \Big[\hat{u}(x_b^i, 0, z_b^i; \boldsymbol{\theta})\Big]^2 \\ \qquad + \Big[\hat{u}(x_b^i, 2, z_b^i; \boldsymbol{\theta})\Big]^2 + \Big[\hat{u}(z_b^i, y_b^i, 0; \boldsymbol{\theta})\Big]^2 + \Big[\hat{u}(x_b^i, y_b^i, 2; \boldsymbol{\theta})\Big]^2, \\ MSE_{\mathcal{F}} = \dfrac{1}{N_{\mathcal{F}}}\displaystyle\sum_{i=1}^{N_{\mathcal{F}}}\Big[\mathcal{F}(x_{\mathcal{F}}^i, y_{\mathcal{F}}^i, z_{\mathcal{F}}^i; \boldsymbol{\theta})\Big]^2. \end{cases} \tag{36}$$

Table 11 Minimum, maximum, mean, and standard deviation for all methods during ten times run in two-dimensional SG equation

Stat	Basic PINN		Basic PINN-TSVD		r-PINN		r-PINN-TSVD	
	Sample	Overall	Sample	Overall	Sample	Overall	Sample	Overall
Min	1.374$e-$04	**1.265e−04**	**1.316e−04**	1.453$e-$04	2.938$e-$04	2.915$e-$04	**2.289e−04**	**2.828e−04**
Max	**2.337e−04**	**2.440e−04**	3.553$e-$04	3.278$e-$04	**5.039e−04**	**4.004e−04**	6.145$e-$04	6.049$e-$04
Mean	**1.803e−04**	**2.036e−04**	2.689$e-$04	2.621$e-$04	**4.077e−04**	**3.532e−04**	4.665$e-$04	4.362$e-$04
Std. Dev.	**3.231e−05**	**4.064e−05**	6.357$e-$05	5.985$e-$05	**6.348e−05**	**3.899e−04**	1.291$e-$04	1.086$e-$04

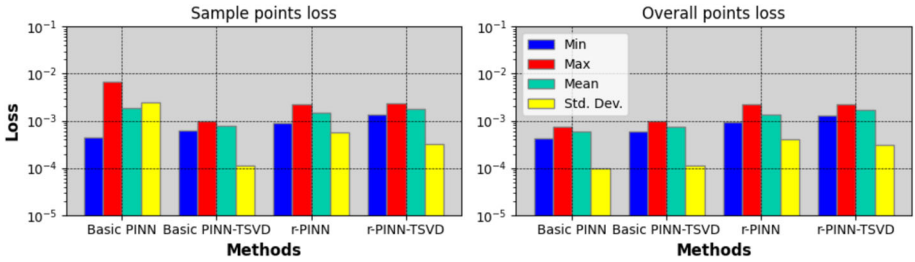


Fig. 25 Visualization of loss function values statistical description from all methods during ten times run in three-dimensional Poisson equation

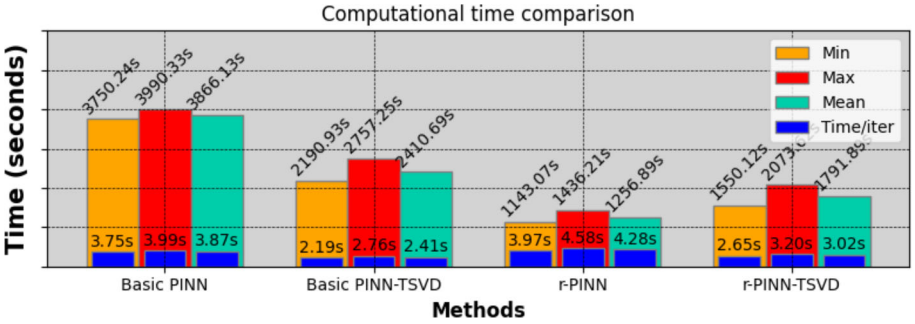


Fig. 26 Comparison of computational performance among all methods in three-dimensional SG equation

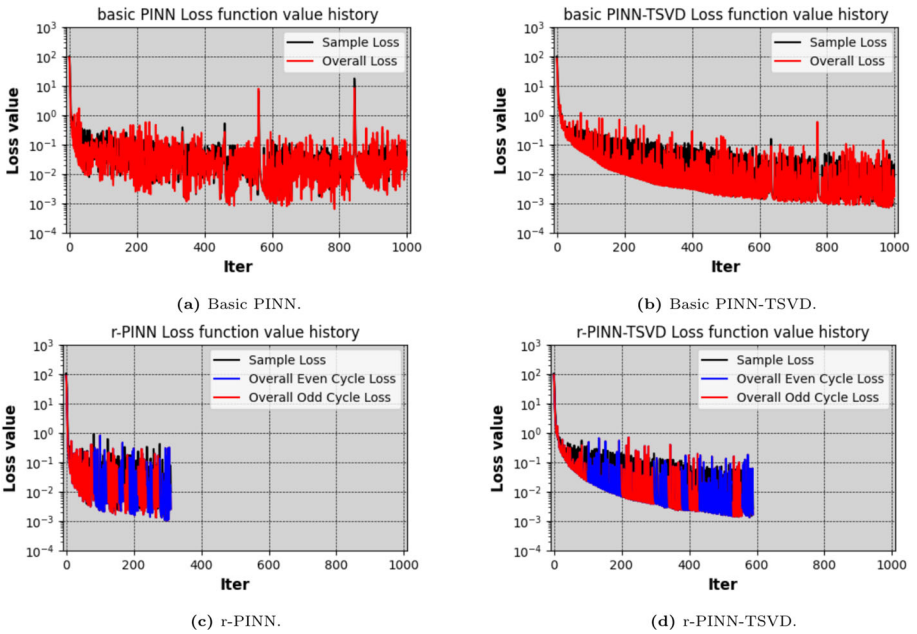
The training process for the three-dimensional SG equation using PINN with a restarting strategy was conducted for 100 iterations, repeated across 10 cycles. Meanwhile, the basic PINN method was performed for 1,000 iterations. The performance of each method across 10 training runs is visualized in Figure 24.

Figure 24 shows that without TSVD, both basic PINN and r-PINN still exhibit slight underfitting or overfitting, as indicated by the discrepancy between sample loss and overall loss. In contrast, TSVD-based methods display more consistent loss values, indicating improved stability. Furthermore, when r-PINN is combined with TSVD, it balances sample and overall loss more effectively, suggesting it may be the most stable and reliable method. The statistical summary of these results is presented in Table 11.

Similar to the previous Poisson results, Table 11 shows that incorporating TSVD yields only a slight improvement in sample loss. These values are further illustrated in Figure 25.

In addition, another advantage of TSVD is illustrated in Figure 26, which shows the computational time comparison among all methods.

Figure 26 shows that the basic PINN method generated an average total training time of 5,043.79 seconds. However, the integration with TSVD significantly reduced the average time to 3,600.50 seconds. This improvement is attributed to a reduction in time per iteration from 5.04 seconds to 3.60 seconds, representing a 28.6% reduction. On the other hand, the r-PINN method achieved an average total training time of 2,425.44 seconds, with an average of 5.08 seconds per iteration. When TSVD is applied, it reduces the average time per iteration by approximately 32.7%, lowering it to just 3.42 seconds. Subsequently, the loss function value history, recorded for up to 1,000 iterations across all methods, is visualized in Figure 27.



(a) Basic PINN. (b) Basic PINN-TSVD.

(c) r-PINN. (d) r-PINN-TSVD.

Fig. 27 Loss function value history for each method in three-dimensional SG equation

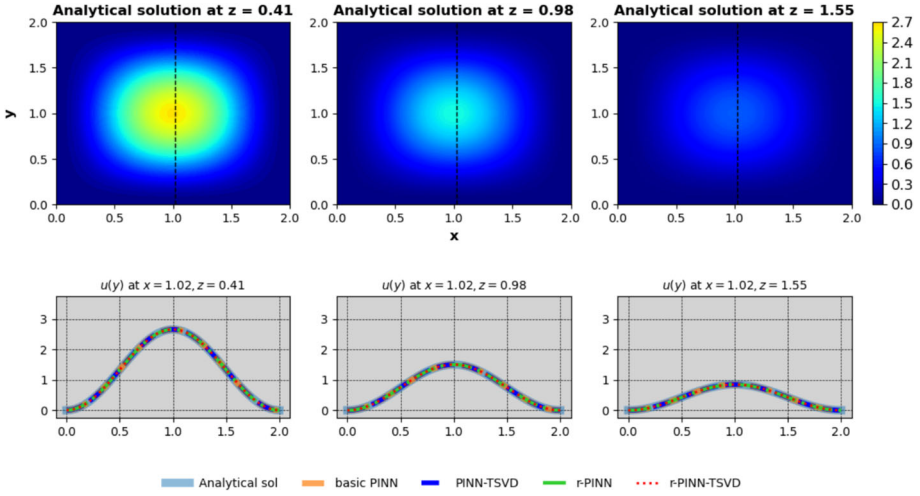


Fig. 28 Analytical solution at $z = 0.41$, $z = 0.98$ and $z = 1.55$ (top) and comparison between the basic PINN, PINN-TSVD, r-PINN, and r-PINN-TSVD solutions for the three-dimensional SG equation at $x = 1.02$ (bottom)

Figure 27 presents a representative result from ten independent runs of the training process. Both the basic PINN and basic PINN-TSVD were trained for 1,000 iterations, whereas r-PINN and r-PINN-TSVD completed training in approximately 500 and 700 iterations, respectively. In terms of loss function history, all methods exhibit smoother patterns during training for this case. This may be attributed to the SG equation, which has smooth spatiotemporal behavior in

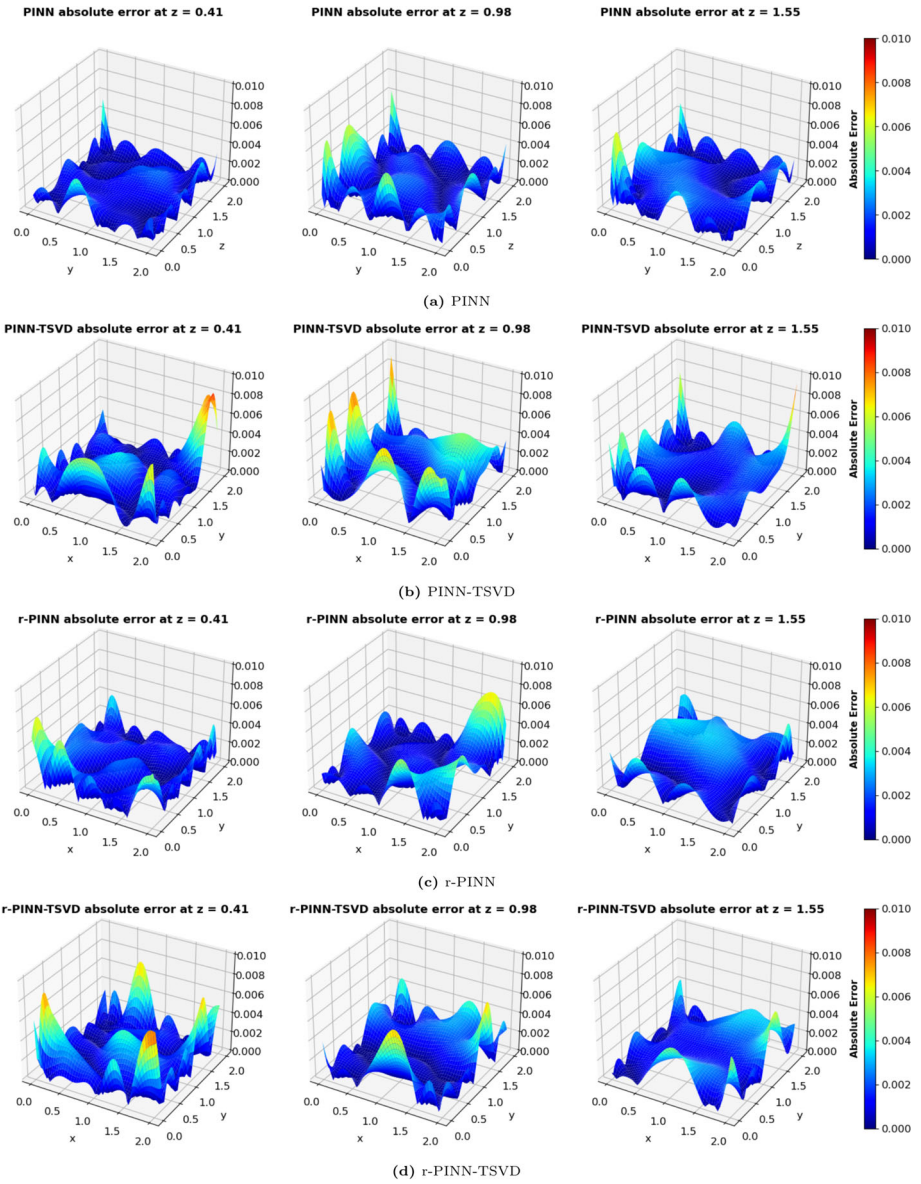
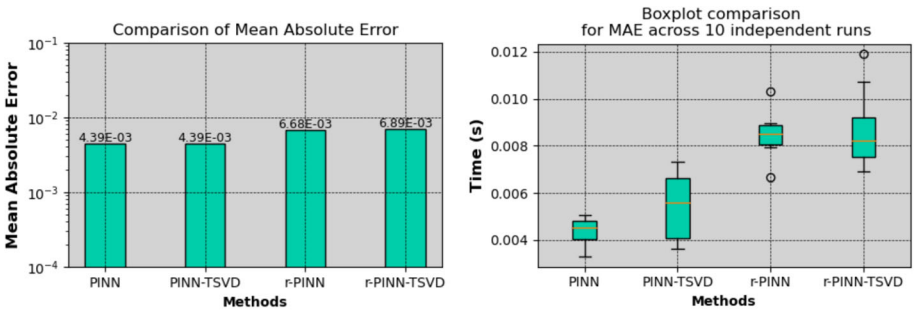


Fig. 29 Comparison of the AE between (a) PINN, (b) PINN-TSVD, (c) r-PINN, and (d) r-PINN-TSVD relative to the analytical solution along the system of SS-NS equations domain

its solution. The approximation results for the three-dimensional SG equation are presented in Figure 28.

The top panel of Figure 28 shows the analytical solution of the three-dimensional SG equation at three cross-sections along the z-axis: $z = 0.41$, $z = 0.98$, and $z = 1.55$. The bottom row compares the predicted profiles at $x = 1.02$ for each z-slice, obtained from basic PINN, PINN-TSVD, r-PINN, and r-PINN-TSVD. All methods closely match the analytical



(a) MAE (b) MAE across 10 independent runs

Fig. 30 Comparison of the MAE between the basic PINN, PINN-TSVD, r-PINN, and r-PINN-TSVD relative to the three-dimensional SG equation's analytical solution

solution, consistent with the trends observed in the Poisson case. Due to minimal visual differences, Figure 29 provides a more detailed AE comparison across the domain at the same z-values.

Overall, Figure 29 shows that all methods tend to produce relatively random error spikes across the domain. However, these distribution plots only visualize the selected spatial slices at $z = 0.41$, $z = 0.98$, and $z = 1.55$, and are not sufficient to determine the overall differences among all methods. Therefore, a comparison of MAE, provided in Figure 30, offers a clearer illustration of each method's overall accuracy relative to the analytical solution.

Figure 30a shows that the basic PINN and PINN-TSVD achieve comparable MAE values, indicating similar accuracy relative to the analytical solution. In contrast, applying the restarting strategy slightly reduces accuracy in this case, as reflected by the higher MAE values for the restarting approaches. The boxplot in Figure 30b further summarizes the MAE distributions across 10 independent runs. Furthermore, Table 12 presents a statistical summary of all the corresponding results.

4.5 Steady-state Navier-Stokes Equations

The final test case for evaluating the proposed methods is the steady-state Navier-Stokes (SS-NS) equations, a fundamental class of SPDEs governing fluid motion. These equations are widely applied in modeling weather patterns [75], ocean currents [76], and aerodynamic flows [77]. The SS-NS system consists of two momentum equations describing velocity behavior under external forces and one continuity equation ensuring fluid incompressibility. Although no closed-form analytical solution exists, grid-based solution data (51×51) from Chiu et al. [62] is used as a benchmark. Figure 31 shows the training points and reference data for $u(x, y)$, $v(x, y)$, and $p(x, y)$.

Given the limited total of 2,601 points, 1,500 collocation points were used, while 100 boundary points were retained and randomly selected from the domain. The ANN architecture in this case was expanded to 5 hidden layers with 50 neurons each. The comparison of the trainable parameters between the TSVD and non-TSVD models for this case is provided in Table 13.

Table 13 also shows the total neurons for the output layer were extended to 3 as it represents the approximate outputs $\hat{u}(x, y)$, $\hat{v}(x, y)$, and $\hat{p}(x, y)$, respectively. Furthermore, the

Table 12 Three-dimensional SG equation summarize table

Method	Cycle	Stat	Total iter	Time/iter	Total time	Loss		MAE
						Sample	Overall	
Basic PINN	1	Min	1000	4.60s	4603.43s	1.374$e-04$	**1.265e−04**	**3.292e−03**
		Max	1000	5.61s	5614.08s	**2.337e−04**	**2.440e−04**	**5.058e−03**
		Mean	1000	4.97s	4970.38s	**1.803e−04**	**2.036e−04**	**4.380e−03**
		Std. Dev.	0	0.29s	289.21s	**3.231e−05**	**4.064e−05**	**5.657e−04**
Basic PINN-TSVD	1	Min	1000	**3.39s**	**3385.21s**	**1.316e−04**	1.453$e-04$	3.613$e-03$
		Max	1000	**3.84s**	**3840.31s**	3.553$e-04$	3.278$e-04$	7.302$e-03$
		Mean	1000	**3.59s**	**3590.75s**	2.689$e-04$	2.621$e-04$	5.448$e-03$
		Std. Dev.	0	**0.16s**	**163.07s**	6.357$e-05$	5.985$e-05$	1.317$e-03$
r-PINN	10	Min	**402**	4.73s	**1900.18s**	2.938$e-04$	2.915$e-04$	**6.680e−03**
		Max	**555**	5.93s	**2891.55s**	**5.039e−04**	**4.004e−04**	**1.033e−02**
		Mean	**471.65**	5.07s	**2394.64s**	**4.077e−04**	**3.532e−04**	**8.470e−03**
		Std. Dev.	46.63	0.34s	300.30s	**6.348e−05**	**3.899e−04**	**8.844e−04**
r-PINN-TSVD	10	Min	572	**3.25s**	1938.24s	**2.289e−04**	2.828$e-04$	6.888$e-03$
		Max	679	**3.67s**	2315.39s	6.145$e-04$	6.049$e-04$	1.190$e-02$
		Mean	630.50	**3.42s**	2153.73s	4.665$e-04$	4.362$e-04$	8.630$e-03$
		Std. Dev.	**33.29**	**0.13s**	**106.95s**	1.291$e-04$	1.086$e-04$	1.546$e-03$

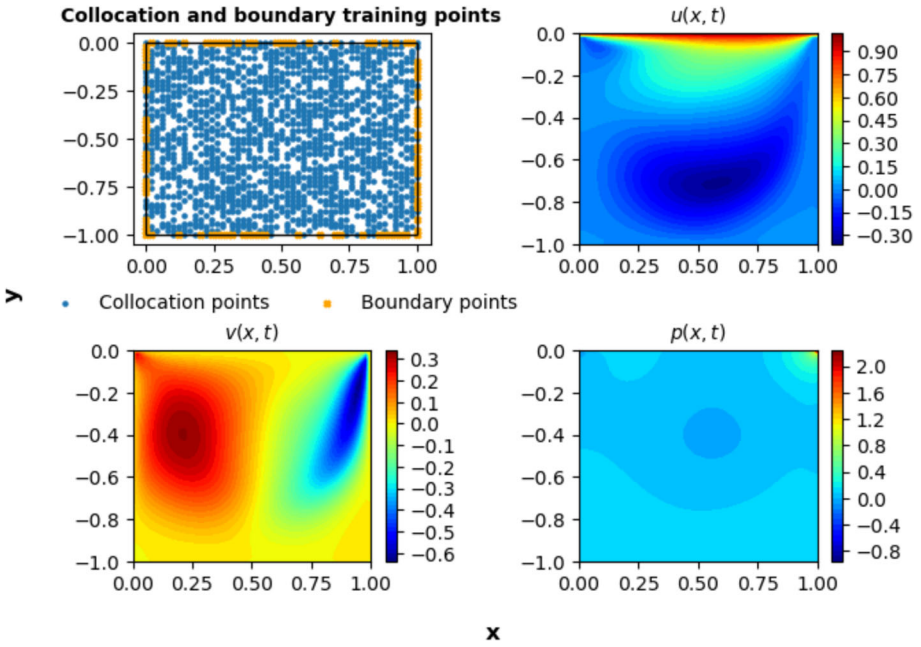


Fig. 31 Randomly selected collocation and boundary training points within the overall domain grid (top-left), ground-truth solution for $u(x, y)$ (top-right), ground-truth solution for $v(x, y)$ (bottom-left), and ground-truth solution for $p(x, y)$ (bottom-right)

Table 13 Trainable parameters comparison between method with TSVD and non-TSVD in the system of SS-NS equations

Layer type	Neurons	Number of trainable parameter	
		With TSVD with $rank = 5$	Without TSVD/basic PINN
Input	2	0	0
Hidden layer 1	50	150	150
Hidden layer 2	50	2,550	2,550
Hidden layer 3	50	2,550	2,550
Hidden layer 4	50	2,550	2,550
Hidden layer 5	50	555	2,550
Output	3	153	153
Total parameters		**8,508**	**10,503**
Size		**159kB**	**180kB**

approximate solution function for this SS-NS case is defined as follows:

$$\begin{cases} \mathcal{F}_1 := \frac{\partial(\hat{u}\hat{u})}{\partial x}(x, y; \theta) + \frac{\partial(\hat{u}\hat{v})}{\partial y}(x, y; \theta) - \frac{1}{Re}\left(\frac{\partial^2 \hat{u}(x,y;\theta)}{\partial x^2} + \frac{\partial^2 \hat{u}(x,y;\theta)}{\partial y^2}\right) + \frac{\partial \hat{p}(x,y;\theta)}{\partial x} \\ \mathcal{F}_2 := \frac{\partial(\hat{u}\hat{v})}{\partial x}(x, y; \theta) + \frac{\partial(\hat{v}\hat{v})}{\partial y}(x, y; \theta) - \frac{1}{Re}\left(\frac{\partial^2 \hat{v}(x,y;\theta)}{\partial x^2} + \frac{\partial^2 \hat{v}(x,y;\theta)}{\partial y^2}\right) - \frac{\partial \hat{p}(x,y;\theta)}{\partial y} \\ \mathcal{F}_3 := \frac{\partial \hat{u}(x,y;\theta)}{\partial x} + \frac{\partial \hat{v}(x,y;\theta)}{\partial y} \end{cases} \tag{37}$$

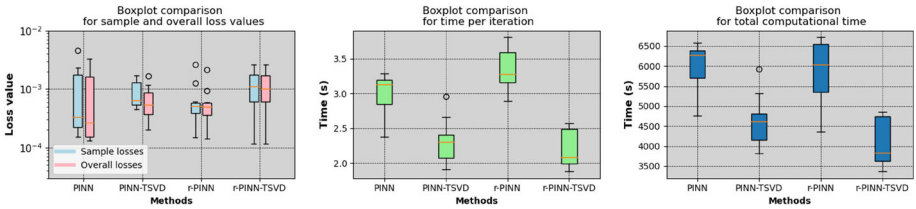


Fig. 32 Boxplot comparison of loss values, time per iteration, total computational time from 10 independent runs between the basic PINN, PINN-TSVD, r-PINN, and r-PINN-TSVD in solving system of SS-NS equations

within the spatial domain $x \in [0, 1]$ and $y \in [-1, 0]$. Here, $\hat{u}(x, y)$ and $\hat{v}(x, y)$ denote the approximations of the velocity components along the x and y-axes, respectively. Additionally, $\hat{p}(x, y)$ represents the pressure distribution within the cavity flow, while the Reynolds number, denoted as Re, is set to $Re = 400$ for this case. The loss function formulation for this case is divided into two separate components, each associated with the main SPDE equations and boundary conditions listed in Table 1. These components are defined as follows:

$$\begin{cases} \mathcal{L}_b(\theta) = MSE_b^1 + MSE_b^2, \\ \mathcal{L}_{\mathcal{F}}(\theta) = MSE_{\mathcal{F}}^1 + MSE_{\mathcal{F}}^2 + MSE_{\mathcal{F}}^3. \end{cases} \tag{38}$$

All MSE terms used in these components are explicitly formulated below:

$$\begin{cases} MSE_b^1 = \frac{1}{N_b}\sum_{i=1}^{N_b}\Big(\big[\hat{u}(0, y_b^i; \theta)\big]^2 + \big[\hat{u}(1, y_b^i; \theta)\big]^2 + \big[\hat{u}(x_b^i, 0; \theta) - 1\big]^2 \\ \quad + \big[\hat{u}(x_b^i, -1; \theta)\big]^2 \Big), \\ MSE_b^2 = \frac{1}{N_b}\sum_{i=1}^{N_b}\Big(\big[\hat{v}(0, y_b^i; \theta)\big]^2 + \big[\hat{v}(1, y_b^i; \theta)\big]^2 + \big[\hat{v}(x_b^i, 0; \theta)\big]^2, \end{cases} \tag{39}$$

$$\begin{cases} MSE_{\mathcal{F}}^1 = \frac{1}{N_{\mathcal{F}}}\sum_{i=1}^{N_{\mathcal{F}}}\Big[\frac{\partial(\hat{u}\hat{u})}{\partial x_{\mathcal{F}}^i}(x_{\mathcal{F}}^i, y_{\mathcal{F}}^i; \theta) + \frac{\partial(\hat{u}\hat{v})}{\partial y}(x_{\mathcal{F}}^i, y_{\mathcal{F}}^i; \theta) \\ \quad - \frac{1}{Re}\left(\frac{\partial^2\hat{u}(x_{\mathcal{F}}^i, y_{\mathcal{F}}^i; \theta)}{\partial (x_{\mathcal{F}}^i)^2} + \frac{\partial^2\hat{u}(x_{\mathcal{F}}^i, y_{\mathcal{F}}^i; \theta)}{\partial (y_{\mathcal{F}}^i)^2} \right) \\ \quad + \frac{\partial\hat{p}(x_{\mathcal{F}}^i, y_{\mathcal{F}}^i; \theta)}{\partial x_{\mathcal{F}}^i} \Big]^2, \\ \quad + \big[\hat{v}(x_b^i, -1; \theta)\big]^2 \Big). \\ MSE_{\mathcal{F}}^2 = \frac{1}{N_{\mathcal{F}}}\sum_{i=1}^{N_{\mathcal{F}}}\Big[\frac{\partial(\hat{u}\hat{v})}{\partial x_{\mathcal{F}}^i}(x_{\mathcal{F}}^i, y_{\mathcal{F}}^i; \theta) + \frac{\partial(\hat{v}\hat{v})}{\partial y_{\mathcal{F}}^i}(x_{\mathcal{F}}^i, y_{\mathcal{F}}^i; \theta) \\ \quad - \frac{1}{Re}\left(\frac{\partial^2\hat{v}(x_{\mathcal{F}}^i, y_{\mathcal{F}}^i; \theta)}{\partial (x_{\mathcal{F}}^i)^2} + \frac{\partial^2\hat{v}(x_{\mathcal{F}}^i, y_{\mathcal{F}}^i; \theta)}{\partial (y_{\mathcal{F}}^i)^2} \right) - \frac{\partial\hat{p}(x_{\mathcal{F}}^i, y_{\mathcal{F}}^i; \theta)}{\partial y_{\mathcal{F}}^i} \Big]^2, \\ MSE_{\mathcal{F}}^3 = \frac{1}{N_{\mathcal{F}}}\sum_{i=1}^{N_{\mathcal{F}}}\left[\frac{\partial\hat{u}(x_{\mathcal{F}}^i, y_{\mathcal{F}}^i; \theta)}{\partial x_{\mathcal{F}}^i} + \frac{\partial\hat{v}(x_{\mathcal{F}}^i, y_{\mathcal{F}}^i; \theta)}{\partial y_{\mathcal{F}}^i} \right]^2. \end{cases} \tag{40}$$

The total loss function $\mathcal{L}(\theta)$ is then formulated as the sum of these two components:

$$\mathcal{L}(\theta) = \mathcal{L}_b(\theta) + \mathcal{L}_{\mathcal{F}}(\theta). \tag{41}$$

Table 14 Minimum, maximum, mean, and standard deviation for all methods during ten times run in system of SS-NS equations

Stat	Basic PINN		Basic PINN-TSVD		r-PINN		r-PINN-TSVD	
	Sample	Overall	Sample	Overall	Sample	Overall	Sample	Overall
Min	**1.523e−04**	**1.321e−04**	4.552e−04	1.995e−04	1.499e−04	1.433e−04	**1.173e−04**	**1.170e−04**
Max	**4.537e−04**	3.297e−03	1.702e−03	**1.670e−03**	**2.590e−04**	**2.150e−03**	2.649e−03	2.612e−03
Mean	1.093e−03	9.004e−04	**8.715e−04**	**6.787e−04**	**7.327e−04**	**6.420e−04**	1.172e−03	1.131e−03
Std. Dev.	1.391e−03	1.092e−03	**4.581e−04**	**4.349e−04**	**6.784e−04**	**5.390e−04**	7.616e−04	7.575e−04

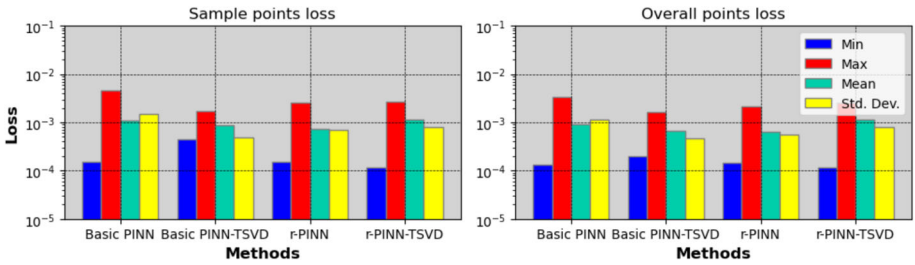


Fig. 33 Visualization of loss function values statistical description from all methods during ten times run in system of SS-NS equations

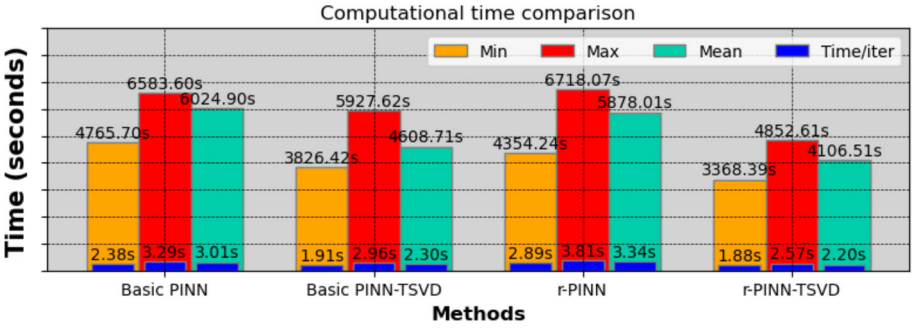
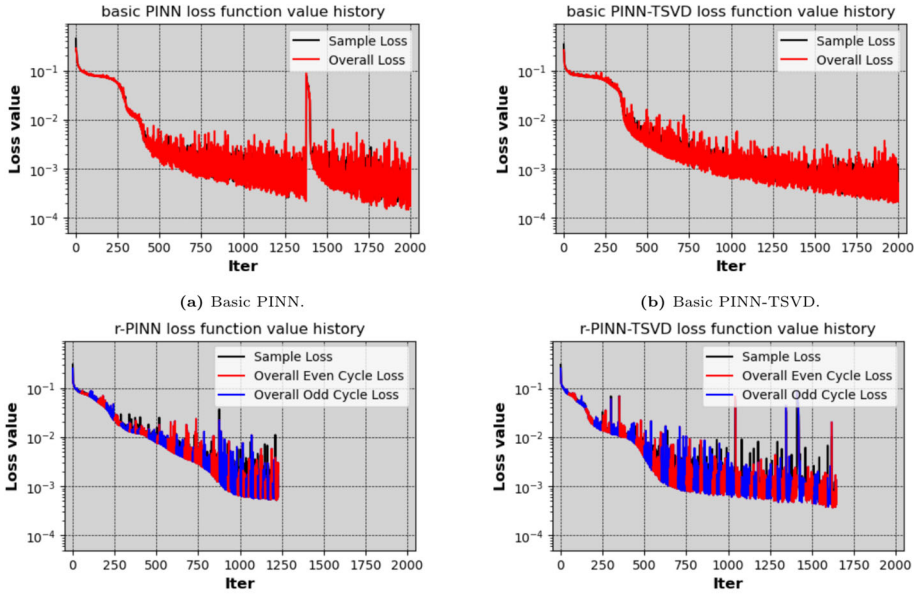


Fig. 34 Comparison of computational performance among all methods in the system of SS-NS equation



(a) Basic PINN.

(b) Basic PINN-TSVD.

(c) r-PINN.

(d) r-PINN-TSVD.

Fig. 35 Loss function value history for each method in system of SS-NS equations

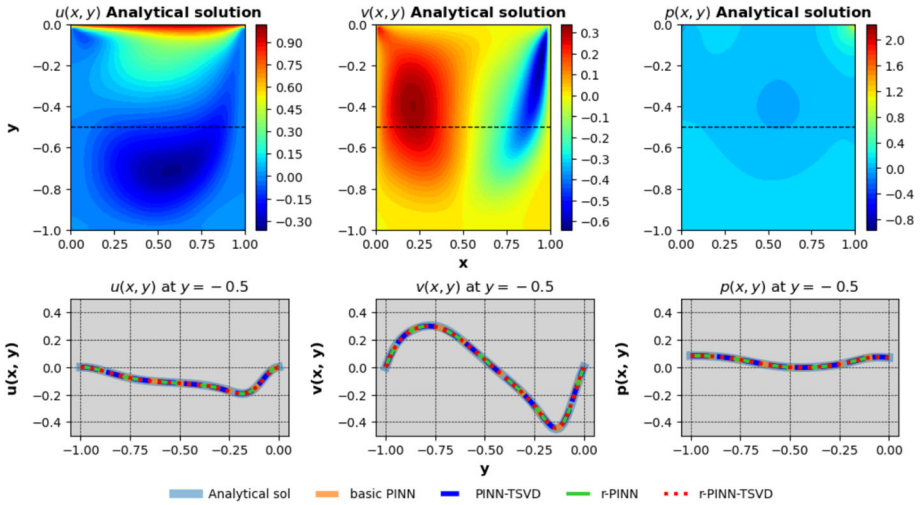


Fig. 36 Analytical solution (top) and comparison between the basic PINN, PINN-TSVD, r-PINN, and r-PINN-TSVD solutions for the SS-NS equation at $y = -0.5$

The loss function is minimized for 50 iterations, repeated over 40 cycles for methods employing a restarting strategy, whereas non-restarting methods perform 2000 iterations in a single sequence. The results from 10 independent training runs for this case are presented in Figure 32.

From Figure 32, it is evident that all methods achieve comparable loss values. Despite this, the methods incorporating TSVD also demonstrate faster time per iteration and total computational time, as evidenced in the middle and right subplots. This suggests that the TSVD models not only achieve faster convergence but also maintain a comparable loss for this case. The statistical summary from the loss function values is presented in Table 14.

As previously discussed, Table 14 shows a relatively balanced outcome between the basic PINN and PINN-TSVD methods. While basic PINN performs better in terms of minimum and maximum loss, PINN-TSVD achieves a lower mean, reflecting improved consistency. However, this pattern does not hold for the restarting variants; r-PINN-TSVD only shows an advantage in the minimum value, indicating a slight lower in overall performance. These values are more clearly visualized in Figure 33.

Furthermore, the computational performance among all methods is presented in Figure 34, showing the advantage of involving TSVD in accelerating the training time per iteration.

Figure 34 shows that for solving a complex problem such as the SS-NS equations, involving TSVD can reduce the average iteration time from 3.01 seconds to just 2.30 seconds, representing a 23.5% reduction. This improvement also leads to a significant decrease in total computational time, reaching 4608.71 seconds. When combined with the restarting approach, the reduction increases to 34.1%, lowering the iteration time from 3.34 seconds to just 2.20 seconds. This improvement in time per iteration also resulted in r-PINN-TSVD achieving the fastest total computational time, completing in only 4106.51 seconds. Furthermore, the loss function history from one of the 10 independent runs is presented in Figure 35.

Figures 35a and 35b show that both the basic PINN and PINN-TSVD required at most 2,000 iterations. These figures also show that a fluctuation occurred around the 1,400th iter-

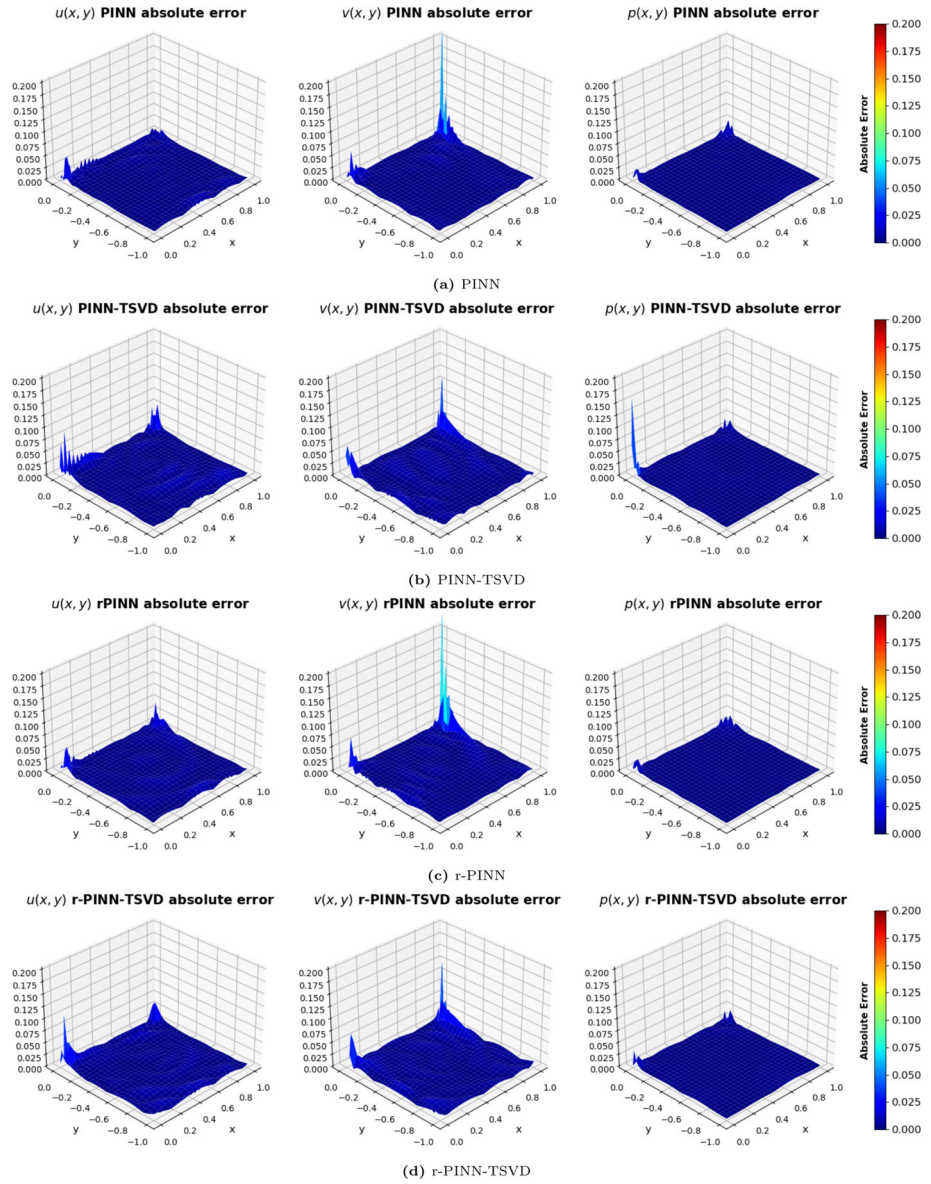
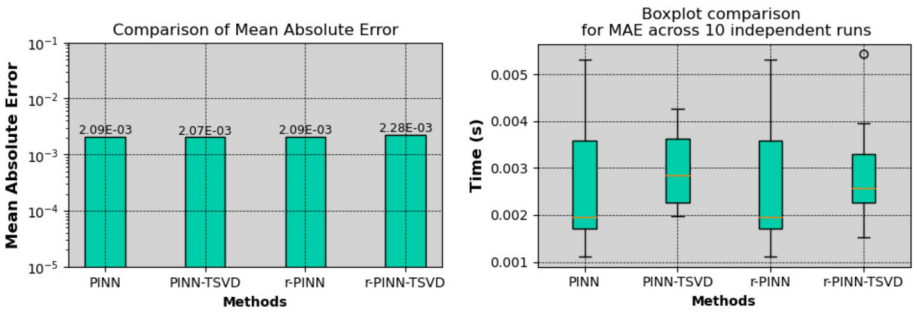


Fig. 37 Comparison of the AE between (a) PINN, (b) PINN-TSVD, (c) r-PINN, and (d) r-PINN-TSVD relative to the analytical solution along the two-dimensional Poisson equation domain

ation during the training process of the basic PINN, while PINN-TSVD exhibited smoother convergence. Figures 35c and 35d

show that one of the r-PINN training runs stopped at around 1,200 iterations, whereas r-PINN-TSVD continued until approximately 1,600 iterations due to the involvement of the restarting strategy. The approximation results for this case are presented in Figure 36.



(a) MAE (b) MAE across 10 independent runs

Fig. 38 Comparison of the MAE between the basic PINN, PINN-TSVD, r-PINN, and r-PINN-TSVD relative to the system of SS-NS equations' analytical solution

Table 15 System of SS-NS equations summarize table

Method	Cycle	Stat	Total iter	Time/iter	Total tim	Loss		MAE
						Sample	Overall	
Basic PINN	1	Min	2000	2.38s	4765.70s	**1.523e−04**	**1.321e−04**	**1.107e−03**
		Max	2000	3.29s	6583.60s	4.537e−03	3.297e−03	5.317e−03
		Mean	2000	3.01s	6024.90s	1.093e−03	9.004e−04	**2.564e−03**
		Std. Dev.	0	**0.26s**	**529.24s**	1.391e−03	1.092e−03	1.340e−03
Basic PINN-TSVD	1	Min	2000	**1.91s**	**3826.42s**	4.552e−04	1.995e−04	1.966e−03
		Max	2000	**2.96s**	**5927.62s**	**1.702e−03**	**1.670e−03**	**4.274e−03**
		Mean	2000	**2.30s**	**4608.71s**	**8.715e−04**	**6.787e−04**	2.988e−03
		Std. Dev.	0	0.31s	617.91s	**4.581e−04**	**4.349e−04**	**8.106e−04**
r-PINN	40	Min	**1485**	2.89s	4354.24s	1.499e−04	1.433e−04	**1.107e−03**
		Max	**1929**	3.81s	6718.07s	**2.590e−03**	**2.150e−03**	**4.844e−03**
		Mean	**1754.40**	3.34s	5878.01s	**7.327e−04**	**6.420e−04**	**2.517e−03**
		Std. Dev.	**125.74**	0.31s	742.15s	**6.784e−04**	**5.390e−04**	1.247e−03
r-PINN-TSVD	40	Min	1777	**1.88s**	**3368.39s**	**1.173e−04**	**1.170e−04**	1.517e−03
		Max	1971	**2.57s**	**4852.61s**	2.649e−03	2.612e−03	5.429e−03
		Mean	1864.4	**2.20s**	**4106.51s**	1.172e−03	1.131e−03	2.897e−03
		Std. Dev.	**56.29**	0.27s	**570.29s**	7.616e−04	7.575e−04	**1.074e−03**

The top row of Figure 36 displays the analytical solutions of $u(x, y)$, $v(x, y)$, and $p(x, y)$. Furthermore, the line plots in the bottom row show the approximate solutions from all tested methods for the SS-NS equation at a specific value of y, compared against the analytical solution. This figure also highlights that all methods show close agreement with the analytical solution in this case. For a clearer evaluation of accuracy, Figure 37 presents the AE distributions for each method.

Another noticeable spike in error across all methods is found near the domain boundaries, particularly for the $v(x, y)$ velocity component. This high spike is likely caused by the difficulty of accurately resolving sharp velocity gradients near the edges. In addition, Figure 38 provides a comparison of the MAE across the evaluated methods.

Figure 38 displays the MAE from a representative run, taken from one of the 10 independent runs shown in Figure 38b . This figure shows that all methods achieve comparable levels of accuracy relative to the analytical solution. Notably, the TSVD-based methods maintain a similar MAE to their non-TSVD counterparts while achieving faster computational time, as previously shown in Figure 34. Figure 38b extends this comparison across 10 independent runs, reaffirming that incorporating TSVD does not compromise accuracy, as the distribution of MAE values remains comparable to those of the baseline methods. Furthermore, Table 15 presents a statistical summary of all the corresponding results.

5 Discussion

In the present study, three PDE problems, including cases up to three dimensions, and one system of PDEs are investigated using an innovative integrated TSVD approach. The approximation results obtained from the integrated TSVD approaches are compared with the analytical solutions and the results from non-TSVD approaches. The following discussion elaborates on the experimental findings.

1. Two sets of simulations, varying the position of TSVD on ANN layers and the truncated TSVD rank, were conducted to determine the most optimal TSVD configuration. The scenario of conducting ten runs is investigated, and it is found that integrating TSVD is most effective when applied to a single layer, specifically the final layer of the network. Moreover, increasing the truncated TSVD rank to 10 or 15 leads to a gradual increase in both time per iteration and total training time. However, the additional computational cost is not accompanied by a significant improvement in accuracy. Thus, applying TSVD at the final layer with a rank of 5 represents a practical choice, balancing computational efficiency and predictive accuracy.
2. A model with three hidden layers and 50 neurons is employed in both the one-dimensional and two-dimensional cases. The integrated TSVD layer, with a rank of 5, has successfully reduced the trainable parameters by up to 45.5% and the model size by up to 42.4%. For the three-dimensional case, the model complexity is increased by incorporating three hidden layers with 200 neurons each. In terms of the trainable parameters, TSVD has successfully reduced the total number of trainable parameters by 46.6%. Furthermore, for the system of PDEs case, the model is extended to five hidden layers, each consisting of 50 neurons. In this case, TSVD reduces the number of trainable parameters from 10,503 to 8,508, resulting in an 18,9% reduction.
3. As a result of reduced model complexity, a shorter computational time is observed, both in time per iteration (up to 41.2%) and in total computation time (up to 37.8%). This observation is supported by Figures 10, 18, 26, and 34.
4. Incorporating TSVD yields favorable results in both the one-dimensional wave equation and the system of SS-NS equations. In the one-dimensional case, TSVD not only reduces training time per iteration but also improves performance, as shown by the loss values and MAE comparisons. For the system of SS-NS equations, TSVD maintains accuracy comparable to its non-TSVD counterparts while still offering computational advantages. In contrast, for the two- and three-dimensional cases, the impact of TSVD is less significant, with improvements restricted to sample-level losses and no significant gains in overall accuracy. These findings suggest that while TSVD provides clear advantages in more complex problems, its impact is limited in cases with simpler spatiotemporal solutions.

6 Conclusion

This study aims to investigate the integration of Truncated Singular Value Decomposition (TSVD) within a layer of a Physics-Informed Neural Network (PINN) that employs a restarting strategy to solve up to three-dimensional Partial Differential Equations (PDEs) and systems of PDEs (SPDEs). The models used in this investigation include a one-dimensional wave equation, a two-dimensional Poisson equation, a three-dimensional sine-Gordon (SG) equation, and the steady-state Navier-Stokes (SS-NS) equations. The accuracy of the proposed method is measured using a Mean Squared Error (MSE)-based loss function and is validated using the Mean Absolute Error (MAE) relative to the analytical solutions of each problem.

Incorporating TSVD has been shown to significantly reduce the number of trainable parameters within the ANN architecture. These results are due to the ability of TSVD to compress the weight matrices into lower-rank matrices that preserve essential features. This reduction also brings significant acceleration in time per iteration during the training process. In more complex cases, such as the wave equation and SS-NS equations, integrating TSVD within the layers of basic PINN and r-PINN not only improves computational performance but also yields superior accuracy compared to the non-TSVD approach, as evidenced by the presented figures and tables. In terms of training, incorporating TSVD also increases the stability of the loss function during the optimization process. This is evidenced by the smoother loss function history provided by the TSVD-based approaches, while their non-TSVD counterparts exhibit more noticeable fluctuations.

However, one limitation in the application of TSVD in this study is that it is only applied to the final layer of the network. This is due to performance issues that arise when multiple TSVD layers are integrated into the network. Increasing the TSVD rank also did not result in a significant increase in loss function values. Consequently, applying TSVD to the final layer with a rank of 5 effectively reduces the number of trainable parameters without significant information loss, thereby yielding superior approximation results.

The proposed approach has wide-ranging applicability, as TSVD can be applied to any ANN layer. It is useful not only for obtaining numerical solutions for PDEs but also for a broader range of general ANN applications. One such application is the ongoing investigation into applying the proposed method to numerical simulations of reaction-diffusion systems. The goal is to optimally control competitive interactions between native and invasive species. PINNs are also compatible with the integration of various complementary techniques. Future work will explore the use of functional-link artificial neural networks and alternative activation functions to further enhance the performance of PINNs.

Acknowledgements This project is funded by Ministry of Higher Education (MOHE) under the Fundamental Research Grant Scheme (FRGS), Ref:FRGS/1/2022/STG06/UMT/03/3. The authors are grateful to ESRC (Grant ES/L011859/1) for partially funding this research.

Author Contributions D.A.P and M.A.B : conceptualization, investigation, methodology, software, validation S.S. and N.M : formal analysis, resources D.A.P : writing original draft M.A.B : writing , review and editing, N.B.I : resources and visualization M.A.B : supervision, project administration A.S: editing, funding acquisition.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Conflicts of Interest The authors declare that they have no conflict of interest.

References

1. Evans G, Blackledge J, Yardley P (2012) Numerical methods for partial differential equations. Springer Science & Business Media, Berlin
2. Gockenbach MS (2005) Partial differential equations: analytical and numerical methods, vol 122 Siam
3. Oane M, Mahmood MA, Popescu AC (2021) A state-of-the-art review on integral transform technique in laser-material interaction: Fourier and non-Fourier heat equations. Materials 14(16):4733
4. Torrent D, Parnell WJ, Norris AN (2018) Loss compensation in time-dependent elastic metamaterials. Phys Rev B 97(1):014105
5. Alshmary RMH (2020) Applications of Partial Differentail Equations, In: Journal of Physics: Conference Series, vol 1591 IOP Publishing, vol 1591, p 012105
6. Song L, Chen S, Wang G (2021) Modeling and analysis of environmental vulnerability based on partial differential equation. Arab J Geosci 14:1
7. Bajalan S, Bajalan N (2021) Novel ANN Method for Solving Ordinary and Time-Fractional Black-Scholes Equation. Complexity 2021:1
8. Ureña F, García Á, Vargas AM (2022) Preface to applications of partial differential equations in engineering. Mathematics 11(1):199 (**MDPI**)
9. Berg J, Nyström K (2018) A unified deep artificial neural network approach to partial differential equations in complex geometries. Neurocomputing 317:28
10. Nüsken N, Richter L (2021) Solving high-dimensional Hamilton-Jacobi-Bellman PDEs using neural networks: perspectives from the theory of controlled diffusions and measures on path space. Partial Differ Equ Appl 2:1
11. Antony ANM, Narisetti N, Gladilin E (2023) FDM data driven U-Net as a 2D Laplace PINN solver. Sci Rep 13(1):9116
12. Géron A (2019) Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems O'Reilly Media,
13. Annervaz K, Chowdhury SBR, Dukkipati A (2018) Learning beyond datasets: knowledge Graph Augmented Neural Networks for Natural language Processing, In: Proceedings of NAACL-HLT, pp 313–322
14. Li S, Song W, Fang L, Chen Y, Ghamisi P, Benediktsson JA (2019) Deep learning for hyperspectral image classification: an overview. IEEE Trans Geosci Remote Sens 57(9):6690
15. Helbing G, Ritter M (2018) Deep Learning for fault detection in wind turbines. Renew Sustain Energy Rev 98:189
16. Guo Y, Cao X, Liu B, Gao M (2020) Solving partial differential equations using deep learning and physical constraints. Appl Sci 10(17):5917
17. Lee H, Kang IS (1990) Neural algorithm for solving differential equations. J Comput Phys 91(1):110
18. Lagaris IE, Likas A, Fotiadis DI (1998) Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans Neural Netw 9(5):987
19. Aarts LP, Van Der Veer P (2001) Neural network method for solving partial differential equations. Neural Process Lett 14:261
20. Aarts LP, Van der Veer P (2001) Solving nonlinear differential equations by a neural network method, In: International conference on computational science Springer, pp 181–189
21. Malek A, Beidokhti RS (2006) Numerical solution for high order differential equations using a hybrid neural network-optimization method. Appl Math Comput 183(1):260
22. Sirignano J, Spiliopoulos K (2018) DGM: a deep learning algorithm for solving partial differential equations. J Comput Phys 375:1339
23. Khan NA, Khalaf OI, Romero CAT, Sulaiman M, Bakar MA (2021) Application of Euler Neural Networks with Soft Computing Paradigm to Solve Nonlinear Problems Arising in Heat Transfer. Entropy 23(8):1053. https://doi.org/10.3390/e23081053

24. Ashfaq A, Sulaiman M, Kumam P, Bakar MA, Miftahudin M (2021) Analysis of a Mathematical Model for Drilling System with Reverse Air Circulation by Using the ANN-BHCS Technique. IEEE Access 9:119188. https://doi.org/10.1109/ACCESS.2021.3107405
25. Khan NA, Sulaiman M, Aljohani AJ, Bakar MA et al (2022) Mathematical models of CBSC over wireless channels and their analysis by using the LeNN-WOA-NM algorithm. Eng Appl Artif Intell 107:104537
26. Raissi M, Perdikaris P, Karniadakis G (2017) Physics informed deep learning (part II): Data-driven discovery of nonlinear partial differential equations. arXiv preprint arXiv:1711.10566
27. Raissi M, Perdikaris P, Karniadakis GE (2017) Physics informed deep learning (part i): data-driven solutions of nonlinear partial differential equations, author=Raissi, Maziar and Perdikaris, Paris and Karniadakis, George Em, arXiv preprint arXiv:1711.10561
28. Raissi M, Perdikaris P, Karniadakis GE (2019) Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, author=Raissi. Maziar and Perdikaris Paris and Karniadakis George E J Comput Phys 378:686
29. Cai G, Li J, Liu X, Chen Z, Zhang H (2023) Learning and Compressing: low-Rank Matrix Factorization for Deep Neural Network Compression. Appl Sci 13(4):2704
30. Chen X, Duan J, Karniadakis GE (2021) Learning and meta-learning of stochastic advection-diffusion-reaction systems from sparse measurements. Eur J Appl Math 32(3):397
31. He Q, Barajas-Solano D, Tartakovsky G, Tartakovsky AM (2020) Physics-informed neural networks for multiphysics data assimilation with application to subsurface transport. Adv Water Resour 141:103610
32. Tartakovsky AM, Marrero CO, Perdikaris P, Tartakovsky GD, Barajas-Solano D (2020) Physics-Informed Deep Neural Networks for Learning Parameters and Constitutive Relationships in Subsurface Flow Problems. Water Resour Res 56(5):e2019WR026731
33. Kissas G, Yang Y, Hwuang E, Witschey WR, Detre JA, Perdikaris P (2020) Machine learning in cardiovascular flows modeling: predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks. Comput Methods Appl Mech Eng 358:112623
34. Kadeethum T, Jørgensen TM, Nick HM (2020) Physics-informed neural networks for solving nonlinear diffusivity and Biot's equations. PLoS ONE 15(5):e0232683
35. Gao H, Sun L, Wang JX (2021) PhyGeoNet: physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain. J Comput Phys 428:110079
36. Ren P, Rao C, Liu Y, Wang JX, Sun H (2022) PhyGeoNet: physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain. Comput Methods Appl Mech Eng 389:114399
37. Winata GI, Cahyawijaya S, Lin Z, Liu Z, Fung P (2020) Lightweight and efficient end-to-end speech recognition using low-rank transformer, In: ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) IEEE, pp 6144–6148
38. Kuchaiev O, Ginsburg B (2017) Factorization tricks for LSTM networks, arXiv preprint arXiv:1703.10722
39. Cahyawijaya S (2021) Greenformers: improving computation and memory efficiency in transformer models via low-rank approximation. Ph.D. thesis
40. Vemuri SK, Büchner T, Niebling J, Denzler J (2024) Functional tensor decompositions for physics-informed neural networks, In: International Conference on Pattern Recognition Springer, pp 32–46
41. Lin J, Wang S, Zhang K, Yan X (2024) Pruning Strategy of Enriched Physics-Informed Neural Network for Two-Phase Flow Simulation in Heterogeneous Porous Media. In: Annual Meeting Conference on Porous Media Springer, pp 1073–1080
42. Pratama DA, Bakar MA, Ibrahim NF, Idris R, Mohamed N (2023) Physical restriction neural networks with restarting strategy for solving mathematical model of thermal heat equation for early diagnose breast cancer. Res Appl Math 19:100384
43. Maharani M, Salhi A (2015) Restarting from Specific Points to Cure Breakdown in Lanczos-type Algorithms. J Math Fundamental Sci 47(2):167
44. Pratama DA, Abo-Alsabeh RR, Bakar MA, Salhi A, Ibrahim NF (2023) Solving partial differential equations with hybridized physic-informed neural network and optimization approach: incorporating genetic algorithms and L-BFGS for improved accuracy. Alex Eng J 77:205
45. Kingma DP, Ba J (2015) Adam: a method for stochastic optimization, In: In: 3rd International conference on learning representations, p 15
46. Lu Y, Yang J (2015) Notes on low-rank matrix factorization, arXiv preprint arXiv:1507.00333
47. Flenner J, Hunter B (2017) A deep non-negative matrix factorization neural network. Semantic Scholar
48. Guo Z, Zhang S (2019) Sparse deep nonnegative matrix factorization. Big Data Mining and Analytics 3(1):13
49. Benjamin Erichson N, Brunton SL, Nathan Kutz J (2017) Compressed singular value decomposition for image and video processing, In: Proceedings of the IEEE International Conference on Computer Vision Workshops, pp 1880–1888

50. Deisenroth MP, Faisal AA, Ong CS (2020) Mathematics for machine. learningCambridge University Press
51. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M et al (2015) TensorFlow: large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/guide/core/matrix_core#svd_fundamentals. Software available from tensorflow.org
52. Goodfellow I, Bengio Y, Courville A, Bengio Y (2016) Deep learning. MIT press Cambridge
53. He K, Zhang X, Ren S, Sun J (2015) Delving deep into rectifiers: surpassing human-level performance on imagenet classification, In: Proceedings of the IEEE international conference on computer vision, pp 1026–1034
54. Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks, In: Proceedings of the thirteenth international conference on artificial intelligence and statistics JMLR Workshop and Conference Proceedings, pp 249–256
55. Saxe A, McClelland J, Ganguli S (2014) Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, In: Proceedings of the International Conference on Learning Represenatations 2014 (International Conference on Learning Represenatations 2014
56. Jagtap AD, Kawaguchi K, Karniadakis GE (2020) Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. J Comput Phys 404:109136
57. Panghal S, Kumar M (2020) Engineering with Computers 1–14
58. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, et al. (2016) Optimization free neural network approach for solving ordinary and partial differential equations, In: 12th $\{USENIX\}$ symposium on operating systems design and implementation ($\{OSDI\}$ 16 265–283
59. Hein KB (2018) Data analysis and machine learning: Using neural networks to solve odes and pdes
60. Panagant N, Bureerat S (2014) Solving partial differential equations using a new differential evolution algorithm, Mathematical Problems in Engineering **2014**
61. Biala T, Jator S (2015) A boundary value approach for solving three-dimensional elliptic and hyperbolic partial differential equations. Springerplus 4(1):1
62. Chiu PH, Wong JC, Ooi C, Dao MH, Ong YS (2022) CAN-PINN: A fast physics-informed neural network based on coupled-automatic-numerical differentiation method. Comput Methods Appl Mech Eng 395:114909
63. Kim D (2019) A Modified PML Acoustic Wave Equation. Symmetry 11(2):177
64. Li J, Feng Z, Schuster G (2017) Wave-equation dispersion inversion. Geophys J Int 208(3):1567
65. Narkhede MV, Bartakke PP, Sutaone MS (2022) A review on weight initialization strategies for neural networks. Artif Intell Rev 55(1):291
66. Yusof NM (2006) Time series modeling and designing and artificial neural network (ann) for revenue forecasting. Ph.D. thesis, Universiti Teknologi Malaysia
67. Aggarwal R, Ugail H (2019) On the Solution of Poisson's Equation using Deep Learning, In: 2019 13th International Conference on Software, Knowledge, Information Management and Applications (SKIMA) IEEE, pp. 1–8
68. Özbay AG, Laizet S, Tzirakis P, Rizos G, Schuller B (2019) Poisson CNN: convolutional neural networks for the solution of the Poisson equation with varying meshes and Dirichlet boundary conditions, arXiv preprint arXiv:1910.08613
69. Xiao X, Zhou Y, Wang H, Yang X (2018) A novel cnn-based poisson solver for fluid simulation. IEEE Trans Visual Comput Graphics 26(3):1454
70. Kazhdan M, Bolitho M, Hoppe H (2006) Poisson surface reconstruction, In: Proceedings of the fourth Eurographics symposium on Geometry processing, vol. 7 , vol. 7
71. Pérez P, Gangnet M, Blake A (2003) Poisson image editing, In: ACM SIGGRAPH 2003 Papers ACM Digital Library, 313–318
72. Griffiths G, Schiesser WE (2010) Traveling wave analysis of partial differential equations: numerical and analytical methods with MATLAB and Maple Academic Press
73. Rigge P (2012) Numerical Solutions to the Sine-Gordon Equation, arXiv: Computational Physics . https://api.semanticscholar.org/CorpusID:116974249
74. Scott AC, Chu FY, McLaughlin DW (1973) The soliton: a new concept in applied science. Proc IEEE 61(10):1443
75. Rohli RV, Li C, Rohli RV, Li C (2021) The Seven Basic Equations in Weather Forecasting Models, Meteorology for Coastal Scientists 171–185
76. Madec G, Delecluse P, Imbard M, Levy C (1997) Ocean general circulation model reference manual, Note du Pôle de modélisation
77. Khairani C, Marpaung T, et al. (2019) Computational analysis of fluid behaviour around airfoil with Navier-Stokes equation, In: Journal of Physics: Conference Series, vol. 1376 IOP Publishing, vol. 1376, p. 012003