International Journal of Non–Linear Mechanics 170 (2025) 104988

Contents lists available at ScienceDirect

International Journal of Non-Linear Mechanics

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

Simulation of 3D turbulent flows using a discretized generative model physics-informed neural networks

Amirhossein Khademi [a,*], Erfan Salari [b], Steven Dufour [a]

[a] Department of Mathematics and Industrial Engineering, Polytechnique Montreal, 2500, Chemin de Polytechnique, H3T 1J4, Montreal, Quebec, Canada
[b] Department of Civil Engineering, Lakehead University, 955 Oliver Rd, P7B 5E1, Thunder Bay, Ontario, Canada

ARTICLE INFO

Dataset link: https://github.com/AmirhosseinnnKhademi/DG-PINN.git

Keywords:
Deep learning
Computational physics
Physics-informed neural networks
Partial differential equations
Navier–Stokes equations
Fluids dynamics

ABSTRACT

Physics-informed neural networks (PINNs) demonstrated efficacy in approximating partial differential equations (PDEs). However, challenges arise when dealing with high-dimensional PDEs, particularly when characterized by nonlinear and chaotic behavior, such as turbulent fluid flows. We introduce a novel methodology that integrates domain discretization, a generative model in the Sobolev function space (H^1), and a gating mechanism to effectively simulate high dimensional problems. The effectiveness of the method, Discretized Generative Model Physics-Informed Neural Networks (DG-PINN), is validated by its application to the simulation of a time-dependent 3D turbulent channel flow governed by the incompressible Navier–Stokes equations, a less explored problem in the existing literature. Domain discretization prevents error propagation by using different neural network models in different subdomains. The absence of initial conditions (IC) in subsequent time steps presents a challenge in identifying optimal network parameters. To address this, discretized generative models are used, improving the model's overall performance. The global solutions' regularity is enhanced compared to previous decomposition techniques by using the H^1 norm of error, rather than L^2. The effectiveness of the DG-PINN is validated through numerical test cases and compared against baseline PINNs and traditional domain decomposition PINNs. The DG-PINN demonstrates improvement in both approximation accuracy and computational efficiency, consistently maintaining accuracy even at later time instances. Moreover, the implementation of a distributed training strategy, facilitated by domain discretization, is discussed, resulting in improved convergence rates and more optimized memory usage.

1. Introduction

Deep learning and scientific machine learning techniques have significantly advanced the field of numerical methods. In particular, artificial neural networks have played a crucial role in identifying and approximating complex nonlinear mechanics, providing powerful tools for solving challenging problems across various scientific domains [1–5]. These contributions span from using neural networks (NN) to accelerate traditional finite element methods (FEM) [6,7], to more purely NN-based approximation approaches. In particular, PINNs demonstrated significant advancements in both time efficiency and accuracy in various tasks, including advanced physics-informed reduced order modeling (ROM) [8,9] and approximating solutions to PDEs. Introduced by Raissi et al. [10,11], PINNs offer an innovative approach that bypasses the necessity for mesh generation and numerical discretization by leveraging automatic differentiation [12]. A key advantage of PINNs is their reduced reliance on labeled data and their incorporation of the underlying physics of the model. This is achieved by integrating the residuals of governing equations into the loss function. Despite notable performance improvements, PINNs face challenges with high-dimensional problems like approximating solutions to the three-dimensional Navier–Stokes equations, including convergence issues and prolonged training times. Ongoing research aims to address these limitations.

Building upon the concept of data clustering [13], domain decomposition techniques, namely conservative PINNs (CPINN) [14] and extended PINNs (XPINN) [15], and parallel PINNs [16] were subsequently introduced. They involve the utilization of separate models for distinct subdomains. This approach enables the adjustment of hyperparameters for each model. To enhance the regularity of the global solution, the average solution at the interface with adjacent subdomains was incorporated in the loss function as an extra loss term.

In a similar research line focusing on the concept of domain discretization by Kharazmi et al. [17,18], a variational approach was incorporated into the PINNs framework to enhance accuracy and reduce training costs. This integration resulted in the development of

* Corresponding author.
E-mail addresses: amirhossein.khademi@polymtl.ca (A. Khademi), Esalari@lakeheadu.ca (E. Salari), steven.dufour@polymtl.ca (S. Dufour).

https://doi.org/10.1016/j.ijnonlinmec.2024.104988
Received 23 October 2024; Received in revised form 2 December 2024; Accepted 8 December 2024
Available online 26 December 2024

non-overlapping decomposed domain models along with projections onto a space composed of high-order polynomials. In this context, the trial space corresponds to the NN space, defined globally across the entire computational domain, while the test space consists of piecewise polynomials. Khademi and Dufour [19] further advanced the PINNs framework by introducing domain decomposition techniques and subdomain transformer networks. These innovations prevented the error propagation and enable the projection of input spaces into higher-dimensional feature spaces, significantly enhancing the ability of PINNs to handle complex multi-physics problems and systems characterized by highly discontinuous solutions.

Another significant challenge faced by PINNs is numerical stiffness, which leads to unbalanced back-propagation gradients during the training process. Extensive efforts have been made to tackle this issue [20–27], primarily employing regularization, penalization, or learning rate annealing techniques. These approaches aim to balance the coefficients of loss terms. However, manually adjusting the coefficients of loss terms is not only time-consuming but also generally inefficient due to the high sensitivity of divergence to these coefficient values. Additionally, these values are problem-specific. To address the challenge of determining optimal penalty coefficients, Wang et al. [20] proposed an adaptive learning rate, drawing inspiration from Kingma and Ba [28], to balance the interplay between the various loss terms. This approach computes the coefficients using a moving average scheme based on the ratio of different loss terms. Nevertheless, the observed enhancement resulting from this method is marginal, particularly in addressing problems with higher dimensions. To expand the concept of such regularization to time-dependent problems, the author introduced the causal PINNs [29] to simulate time-dependent dynamical systems exhibiting multi-scale, chaotic, or turbulent behavior. In the causal PINNs framework, loss terms are multiplied by weighting coefficients, allowing a specific loss term $L_r(t_i, \theta)$ to be minimized only if $L_r(t_k, \theta)_{k=1}^{i}$ before t_i are properly minimized. Song et al. [30] recently introduced LA-PINN, a dynamic and point error-based weighting model. LA-PINN utilizes independent loss-attentional networks for each loss component, dynamically updating weights for individual training points during the training process, to improve convergence in stiff regions.

Inspired by the concepts of the XPINN [15] and the causal PINNs [29], Penwarden et al. [31] proposed a unified framework aimed at describing various causality-enforcing PINNs techniques, specifically applicable to time-dependent PDE. This framework not only encompasses existing methods but also provides a platform for developing new causality-enforcing techniques. By leveraging this unified framework, they successfully reduced training times, enhancing the scalability of PINNs on problems that do not pose significant training challenges. To resolve the issue of extended training time, Psaros et al. [32] proposed to incorporate a metalearning approach within PINNs. This work focused on enhancing the baseline PINNs' performance for out-of-distribution tasks and offered a more efficient allocation of computational resources during training. The application of this method to real-world problems is yet to be explored. In a similar work, Penwarden et al. [33] conducted a review of model-agnostic metalearning and transfer learning principles, then applied a model-aware adaptation of metalearning to PINNs. The proposed approach is tested on various canonical forward-parameterized PDEs, demonstrating the potential for accelerating optimization in PINNs.

Early deep neural networks (DNN) and PINNs models were mostly limited to generic and canonical problems [10,11,14,15,17,20,34]. Recent progress has enabled the study of more realistic scenarios through the use of increasingly sophisticated models. Biswas and Anand [35] applied PINNs to simulate three-dimensional laminar flow governed by the Navier–Stokes equations. Their work demonstrated the capability of PINNs to handle complex three-dimensional fluid dynamics problems without relying on traditional numerical methods. Hu et al. [36] presented an innovative approach that combines a physics-informed neural network with a characteristic-based split algorithm to solve Navier–Stokes equations more efficiently. This method reduces the need for extensive labeled data, cuts down memory usage and computational time, and improves the accuracy and convergence of solutions for 3D incompressible flows. Hijazi et al. [37] introduce a Reduced Order Model (ROM) combining POD-Galerkin methods and PINNs to address inverse problems in the Navier–Stokes equations efficiently. Their approach enhances computational efficiency and accuracy by integrating physical laws directly into the loss functions of deep learning models. Cho et al. [38] developed a separable physics-informed neural network (SPINN) that optimizes the solving of multi-dimensional PDEs by minimizing computational and memory needs, thus facilitating quicker and more precise solutions on regular GPUs. Anagnostopoulos et al. [39] introduce a residual-based attention mechanism for PINN framework, which accelerates convergence by focusing on high-error regions without additional computational costs. Their method significantly improves the accuracy and speed of solving PDEs by dynamically adjusting the training focus, leading to an order of magnitude faster convergence in benchmark problems. Jin et al. [40] develop the Fourier Warm Start (FWS) algorithm to improve PINNs by balancing convergence across frequencies and reducing spectral bias. Their Fourier Analysis Boosted PINN (Fab-PINN) shows substantial performance gains, notably reducing L^2 errors and accelerating convergence in complex PDE scenarios. Wang et al. [41] introduce stacked deep learning models for quickly approximating steady-state Navier–Stokes equations at low Reynolds numbers. Their weakly supervised approach captures boundary and geometric conditions without requiring labeled data, significantly reducing computational demands and enabling fast, high-quality fluid flows simulations.

The study of real-world high-dimensional systems typically present the challenge of turbulence, necessitating sophisticated models to accurately analyze fluid dynamics and Navier–Stokes equations. Beck et al. [42] proposed a novel data-based technique to derive large eddy simulations (LES) subgrid closure terms using a deep neural networks model. Zhao et al. [43] introduced a novel computational fluid dynamics (CFD)-based machine learning framework tailored for the development of Reynolds-averaged Navier–Stokes (RANS) models. This approach utilizes CFD simulations to inform and refine machine learning algorithms, enhancing the accuracy of RANS modeling. To showcase the efficacy of this method, it was employed in the context of developing models for wake mixing in turbomachines. Notably, the application of the CFD-driven machine learning framework led to significant improvements in the predicted wake mixing profiles. Zhang et al. [3] later proposed a bi-fidelity shape optimization method for turbulent fluid-flow applications, integrating LES and RANS models within a hierarchical-Kriging surrogate framework. The model enhanced RANS models to capture LES behavior, improving correlation across the design space. Demonstrating efficacy on the periodic-hill case, this approach converged to the LES-optimum with just two LES samples, outperforming standard RANS models. Jin et al. [25] used PINNs to simulate 3D time-dependent turbulent channel flow at $Re \approx 1000$ without a turbulence model. They investigated loss function weighting for data and physics balance, achieving good agreement with DNS outcomes but with high training time. However, it was achieved at a notably high training time. Using a similar direct modeling strategy, Hanrahan et al. [44] utilize physics-informed neural networks (PINNs) with sparse data to effectively model turbulent flows. Their approach, which does not use turbulence models, enables the network to infer key dynamics like Reynolds-stress fields. This study shows that PINNs can robustly predict complex turbulent behaviors using limited experimental data.

Given the literature review, using PINNs and their extensions for solving the Navier–Stokes equations has demonstrated prospective results. Specifically, enhancing the performance of the baseline PINNs through decomposition strategies, metalearning, and transfer learning shows promise. However, significant computational cost, prolonged

training time, challenging solution regularity, and high approximation error in complex phenomena in high dimensional time-dependent PDEs have constrained the applicability of these methods to generic problems or restricted their use to very small computational domains in realistic high-dimensional fluid flow problems.

In this study, a novel domain decomposition technique for the PINNs is introduced. This technique involves discretizing the global domain into multiple overlapping subdomains, allowing for parallel solutions of each subdomain. A generative approach in the Sobolev function space is introduced such that subdomains predict and propagate new training data to be used as IC in the subsequent subdomains. A gating mechanism is implemented to control the interconnection between parallel models, permitting optimal global training. To extend the application scope of PINNs beyond generic problems, we have verified this approach using a real-world problem involving turbulent channel flow governed by the 3D time-dependent Navier–Stokes equations. The results demonstrate that approximations by the DG-PINN closely align with the reference solution. Moreover, comparative numerical analyses reveal that DG-PINN improves both training time and convergence rates. Notably, DG-PINN mitigates the likelihood of solution divergence, a challenge commonly encountered in baseline PINNs, especially in scenarios involving 3D turbulent fluid flow problems, where manual adjustment of loss terms coefficients is crucial to prevent divergence. Below are the key contributions of this study:

- Introduction of a novel domain discretization technique for PINNs aimed at reducing computational cost and enhancing scalability for high-dimensional turbulent fluid flow problems.
- Implementation of a gating mechanism to control the interaction between parallel models dynamically, ensuring accuracy while minimizing computational overhead.
- Introduction of a generative modeling approach within PINNs to optimally approximate and transfer initial conditions to subsequent models, thereby reducing approximation error in subsequent time steps where the problem lacks initial conditions, thereby enhancing approximation's accuracy.
- Enhancing the physics-informed part of PINNs by leveraging the H^1 norm of error to effectively promote solutions' regularity.
- Implementation of a distributed training strategy.
- Expanding the applicability of existing domain decomposition PINN methods: The proposed approach is validated through a real-world application involving turbulent channel flow governed by the 3D time-dependent Navier–Stokes equations, demonstrating the potential of PINNs to tackle complex fluid flow problems.

2. Methodology

2.1. Neural networks

According to the universal approximation theorem, a multi-layer perceptron having only one hidden layer and a limited number of neurons can accurately approximate any continuous function to an arbitrary degree of precision [45–47]. In this study, we approximate solutions to the Navier–Stokes equations using multiple deep sequences of interconnected neurons, structured in a fully connected feed-forward (FFNN) architecture, formulated as

$$f_i(\boldsymbol{x}_i;\boldsymbol{\omega}_i,\boldsymbol{b}_i) = \alpha_i(\boldsymbol{\omega}_i \cdot \boldsymbol{x}_i + \boldsymbol{b}_i) \tag{1}$$

where $\boldsymbol{x}$ is the input, $\boldsymbol{\omega}$ denotes the weight vector, $\boldsymbol{b}$ represents the bias vector and α is a non-linear activation function. An FFNN is a type of architecture of artificial neural network where the connections between its neurons are structured so that each neuron in one layer is connected to every neuron in the subsequent layer. "Fully connected" implies that all neurons from one layer are connected to every neuron in the following layer without skipping any connections. Information flows from the input layer, through one or more hidden layers, and finally to the output layer without any feedback loops or cycles, hence the term "feed-forward". In this network architecture, each neuron's output is determined by applying a weighted sum of the inputs from the preceding layer, followed by an activation function that introduces non-linearities to the network. These activation functions enable the neural network to learn and model complex relationships within the data. Feed forward neural networks are foundational in machine learning and are used in various applications, including pattern recognition, regression, classification, and more complex tasks such as natural language processing, image recognition, and approximation and identification of differential equations.

In our designated NN model to approximate solutions to the Navier–Stokes equations, spatiotemporal coordinates, $\boldsymbol{X} = (t, x, y, z)$, are given to the neural network while instantaneous velocity and pressure fields, $\boldsymbol{U}(\boldsymbol{X}) = (u, v, w)$ and $P(\boldsymbol{X})$, are directly approximated as the model's output, such that

$$\boldsymbol{U}_\theta(\boldsymbol{X}), P_\theta(\boldsymbol{X}) = f_{NN}(\boldsymbol{X}, \theta). \tag{2}$$

The neural network function is denoted by $f_{NN}(\boldsymbol{X}, \theta)$, where θ represents the parameters (weights and biases) of the neural network. The training involves adjusting the parameters θ of the neural network to minimize the difference between the predicted solutions $\boldsymbol{U}_\theta(\boldsymbol{X})$ and the reference solutions $\boldsymbol{U}(\boldsymbol{X}) = (u, v, w)$.

2.2. PINN

Physics-Informed Neural Networks are a class of machine learning models that leverage both data and known physical laws to make predictions. They have gained popularity in scientific computing and engineering for their ability to incorporate prior knowledge of underlying physics into the learning process. PINNs offer a promising approach for solving inverse problems, system identification, and modeling complex physical systems. Let $u(x, t)$ represent the solution to a physical system, where $x \in \Omega \subset \mathbb{R}^d$ denotes the spatial coordinates and $t \in [0, T]$ denotes time. The governing equation describing the system can be written as:

$$\frac{\partial u}{\partial t} + \mathcal{N}(u; x, t) = 0, \tag{3}$$

where $\mathcal{N}(u; x, t)$ represents the nonlinear operator governing the dynamics of the system. PINNs aim to learn the solution $u(x, t)$ directly from data while satisfying the physics described by Eq. (3). This is achieved by minimizing a loss function consisting of two components: a data fidelity term and a physics-informed regularization term. The data fidelity term penalizes the discrepancy between the predicted solution and observed data, typically defined as:

$$L_{\text{data}} = \sum_{i=1}^{N_{\text{data}}} \|u(x_i, t_i) - u_{\text{obs}}(x_i, t_i)\|^2, \tag{4}$$

where N_{data} is the number of data points, (x_i, t_i) are the spatial–temporal coordinates of the data points, and $u_{\text{obs}}(x_i, t_i)$ are the corresponding observed values.

The physics-informed regularization term enforces the satisfaction of the governing PDE in Eq. (3). This is typically formulated as:

$$L_{\text{physics}} = \sum_{j=1}^{N_{\text{phys}}} \|\mathcal{N}(u(x_j, t_j); x_j, t_j)\|^2, \tag{5}$$

where N_{phys} is the number of physics-informed points sampled from the domain. The overall loss function for training the PINNs model is given by:

$$L_{\text{PINN}} = L_{\text{data}} + L_{\text{physics}}, \tag{6}$$

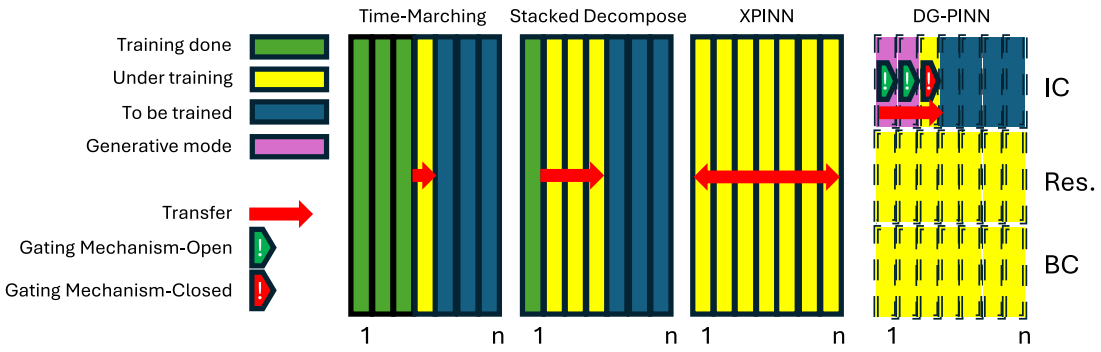

Fig. 1. Illustration of existing decomposition methods compared with the DG-PINN.

2.3. Related works

XPINN: Jagtap and Karniadakis [15] proposed an extension to the CPINN [14] to decompose spatio-temporal domain into subdomains and using independent models in each subdomain that are linked to adjacent subdomain through average solution at the interface, such that:

$$u_{average} = \frac{u_1 + u_2}{2}. \tag{7}$$

In this settings, u_1 and u_2 are the solutions at the interface by two independent adjacent models. The continuity of the global solution is then preserved by enforcing $MSE(u_{average} - u_1) + MSE(u_{average} - u_2) = \|u_{average} - u_1\|_2^2 + \|u_{average} - u_2\|_2^2$. This approach lacks causality and encounters training challenges similar to those seen in standard PINNs. These issues can be exacerbated when dealing with interfaces and separate networks, as they pose a more complex optimization problem by disregarding the causal structure during the training process for time-dependent PDEs [29]. To elaborate, the effectiveness of transfer learning for subsequent models relies on the prior convergence of solutions from previous time steps.

Stacked Decomposition: Penwarden et al. [31] expanded the concept of the XPINN to make it more applicable on time-dependent PDE. Using the causality in PINNs [29], causal weights were applied to the loss function of subdomains to take the significance of causality into account. In this settings, the loss function of the PINNs is re-written as below:

$$L_r(\theta) = \frac{1}{N_t}\sum_{i=1}^{N_t} exp\big(-\epsilon \sum_{k=1}^{i-1} L_r(t_k, \theta)\big) L_r(t_i, \theta) \tag{8}$$

where ϵ is the causality hyperparameter. Consequently, the $L_r(t_i, \theta)$ will be minimized only if the previous residuals $L_r(t_k, \theta)_{k=1}^{i-1}$ are minimized to a predefined value. Additionally, the stacked-decomposition method is characterized by two parameters, namely n and dS. The duration covered by each subdomain over time is determined based on the total time domain of the problem and the number of partitions n. When $dS = 1$, the stacked-decomposition process occurs sequentially rather than concurrently. Conversely, when $dS = n$ and using XPINN interface conditions with all domains activated at the onset of training, the stacked-decomposition method is equivalent to the conventional XPINN approach. The illustrations depicting the decomposition of the XPINN, the Stacked-Decomposition, and the DG-PINN, as described in the following section, are presented in Fig. 1. Also, Table 1 represents a brief specification of these methods and highlights their differences and contributions.

2.4. Discretized generative model physics-informed neural network

The domain discretization technique involves decomposing the domain into multiple overlapping subdomains and using separate models with subdomain-specific hyperparameters and parallel training. This approach enables the representation of the entire domain by combining these subdivided regions as

$$\Omega = \cup_{i=1}^{N_t} \Omega_i \ , \tag{9}$$

where

$$\Omega_i \cap \Omega_j = \Omega_{ij} \ , \tag{10}$$

where N_t shows the total number of partitioned subdomains and Ω_{ij} is the overlapping zone between any two adjacent subdomains. Within this context, the approximation of the solution within the i_{th} subdomain by the i_{th} model is

$$\boldsymbol{u}_{\theta_i} = f_{NN}^{i}(\boldsymbol{x}, \boldsymbol{\theta}_i), \tag{11}$$

where $\boldsymbol{u}_{\theta_i}$ is the latent solution and f_{NN}^{i} is the neural network model in the i_{th} subdomain, with parameters $\boldsymbol{\theta}_i$. A problem, involving 3D time-dependent turbulent channel flow is considered. This flow is governed by the incompressible Navier–Stokes equations which are written in the VP form as

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u} + \nabla p - \frac{1}{Re}\Delta\boldsymbol{u} = 0 \qquad \text{in } \Omega; \tag{12}$$

$$\nabla\cdot\boldsymbol{u} = 0 \qquad \text{in } \Omega; \tag{13}$$

$$\boldsymbol{u} = u_\Gamma \qquad \text{on } \Gamma_D; \tag{14}$$

$$\frac{\partial \boldsymbol{u}}{\partial n} = 0 \qquad \text{on } \Gamma_N, \tag{15}$$

where the non-dimensional time is denoted by t. $\boldsymbol{u}(x, y, z, t) = [u, v, w]^T$ refers to the non-dimensional velocity vector, while P represents the non-dimensional pressure. The term Re stands for the Reynolds number, which characterizes the flow by comparing inertial forces with viscous forces. Reynolds number is defined as $Re = \rho\ u\ L/\mu$, where ρ signifies fluid density, u is the characteristic velocity, L denotes the characteristic length scale, and μ represents dynamic viscosity. The Dirichlet and Neumann boundaries are provided by (14) and (15), respectively.

In this study, we are interested in approximating solutions to the Navier–Stokes equations, using multiple neural networks, novel physics-informed parts, and a distributed optimization strategy based on a mini-batch gradient descent approach.

For the VP form of the time-dependent Navier–Stokes Eqs. (12) to (15), the residuals of the PDEs, including residuals of the momentum equation and the divergence-free constraint, could be written as

$$R_x = \partial_t u + u\partial_x u + v\partial_y u + w\partial_z u + \partial_x p - \frac{1}{Re}(\partial_{xx}^2 u + \partial_{yy}^2 u + \partial_{zz}^2 u) \tag{16}$$

$$R_y = \partial_t v + u\partial_x v + v\partial_y v + w\partial_z v + \partial_y p - \frac{1}{Re}(\partial_{xx}^2 v + \partial_{yy}^2 v + \partial_{zz}^2 v) \tag{17}$$

$$R_z = \partial_t w + u\partial_x w + v\partial_y w + w\partial_z w + \partial_z p - \frac{1}{Re}(\partial_{xx}^2 w + \partial_{yy}^2 w + \partial_{zz}^2 w) \tag{18}$$

$$R_c = \partial_x u + \partial_y v + \partial_z w \tag{19}$$

where R_x, R_y, R_z, and R_c refers to the residuals of the momentum equations in x, y, and z directions and divergence free constraint, respectively. For computations of partial differential operators automatic

Table 1
Classification and comparison of related methods with the DG-PINN.

Method	Soft causality	Hard causality	Function space	Parallel computing	Reported application
XPINN	–	–	L^2	parallel	1D Burgers eq.
Stacked-Decomposition	Weights	Time marching	L^2	Partially parallel	Convection eq. Allen–Cahn eq. Korteweg–de Vries eq.
DG-PINN	Gating	Generative modeling	H^1	parallel	3D time-dependent Navier–Stokes (Turbulent)

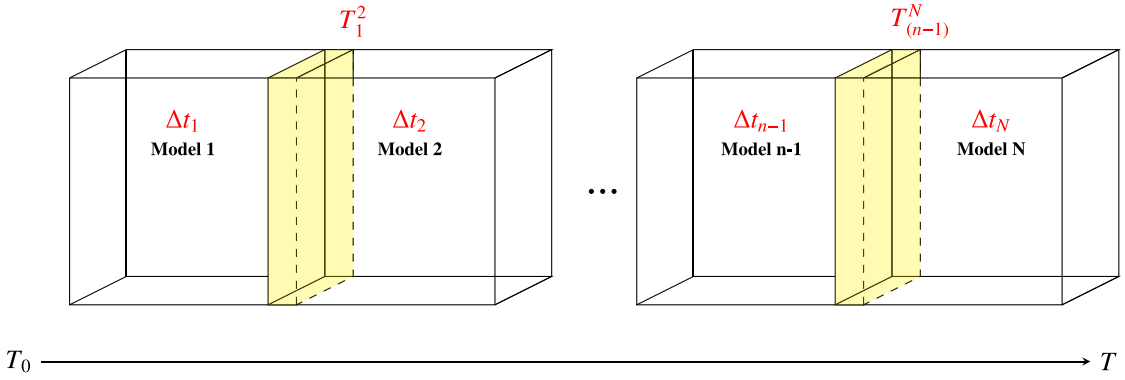

Fig. 2. Schematics of the model discretization.

differentiation [12] is used. To approximate the derivatives in the governing equations, the derivatives of the outputs of the computational graph are calculated with respect to x, y, z, and t.

In PINNs settings, the approximation problem is transformed into an optimization problem of the network parameters $\boldsymbol{\theta}$, where minimization of the function associated with the approximation yields optimal solutions. The associated function is called the loss function which is written as

$$L = L_{IC} + L_{BC} + L_R \tag{20}$$

and the loss terms are written as

$$L_{\mathrm{IC}} = \frac{1}{N_{\mathrm{I}}} \sum_{n=1}^{N_{\mathrm{I}}} \left| \boldsymbol{u}_\theta^n - \boldsymbol{u}_{\mathrm{I}}^n \right|^2 \tag{21}$$

$$L_{\mathrm{BC}} = \frac{1}{N_{\mathrm{B}}} \sum_{n=1}^{N_{\mathrm{B}}} \left| \boldsymbol{u}_\theta^n - \boldsymbol{u}_{\mathrm{B}}^n \right|^2 \tag{22}$$

$$L_{\mathrm{R}} = \frac{1}{N_{\mathrm{R}}} \left(\sum_{n=1}^{N_{\mathrm{R}}} \left| R_x^n \right|^2 + \sum_{n=1}^{N_{\mathrm{R}}} \left| R_y^n \right|^2 + \sum_{n=1}^{N_{\mathrm{R}}} \left| R_z^n \right|^2 + \sum_{n=1}^{N_{\mathrm{R}}} \left| R_c^n \right|^2 \right), \tag{23}$$

where L_{IC}, L_{BC}, and L_R correspond to the error associated with the approximations of the IC, BC, and residuals of the governing PDEs, respectively.

When applying the baseline PINNs approach to time-dependent PDEs, optimizing the network becomes challenging. Although capturing all the chaotic behavior of the PDE in different time steps with a single model is not impossible based on the universal approximation theorem [45–47], this becomes highly time-consuming and prone to divergence. Furthermore, the absence of IC compared to the evolving constraints across the time domain, such as boundary conditions (BC) and PDE residuals, results in noticeable approximation errors in subsequent time steps.

Here we put a novel decomposition technique into practice. The temporal domain is discretized such that there is an overlapping zone between any two adjacent subdomains (the zones highlighted in yellow in Fig. 2). Since the IC is provided solely within the first subdomain, the subdomain's model approximates the latent solution at the spatial coordinates corresponding to the IC, but at the time step corresponding to the overlapping zone. With this generative model, the neural network outputs from one model are used as initial conditions in the subsequent model for training (Represented by the red arrow in Fig. 3). In this setting, the loss term associated with the approximation of IC in the subsequent model is written as

$$L_{\mathrm{IC}}(\boldsymbol{X}) = \frac{1}{N_{\mathrm{I}}} \sum_{n=1}^{N_{\mathrm{I}}} \left(\| \boldsymbol{u}_{\theta_i}^n - \boldsymbol{u}_{\theta_{i-1}}^n \|_{L^2}^2 + \| \nabla (\boldsymbol{u}_{\theta_i}^n - \boldsymbol{u}_{\theta_{i-1}}^n) \|_{L^2}^2 \right) \tag{24}$$

which is equivalent to:

$$L_{\mathrm{IC}}(\boldsymbol{X}) = \frac{1}{N_{\mathrm{I}}} \sum_{n=1}^{N_{\mathrm{I}}} \| \boldsymbol{u}_{\theta_i}^n - \boldsymbol{u}_{\theta_{i-1}}^n \|_{H^1}^2 \tag{25}$$

$$\boldsymbol{X} = (x_0, y_0, z_0, t_{int}) \tag{26}$$

where the subscript 0 denotes the correspondence to the IC, t_{int} represents the time step corresponding to the overlapping zone, $\boldsymbol{u}_{\theta_i}^n$ is the approximation to the IC in the receiving subdomain, and $\boldsymbol{u}_{\theta_{i-1}}^n$ is the predicted solution at the overlapping zone (to serve as training data), generated by the transmitter subdomain

$$\boldsymbol{u}_{\theta_{i-1}} = f_{NN}^{i-1}(\boldsymbol{X}, \boldsymbol{\theta}_{i-1}). \tag{27}$$

The process involves sequentially transferring approximations to be used as IC from the first to the last subdomains. This transfer mechanism ensures a systematic propagation of IC throughout all the subdomains. The schematics of the process is illustrated in Fig. 1. In this figure, the generative mode indicates that the subdomain has reached the accuracy threshold, signaling the opening of the gating mechanism. This allows for the transfer of training data to the subsequent subdomain while also continuing simultaneous training.

The term "Generative" is used in this work to describe the iterative process by which the model generates new approximations at temporal points or coordinates where no training data were originally provided. Unlike traditional supervised learning, which relies solely on labeled data, this approach enables the model to autonomously create approximations in previously unexplored regions of the temporal domain. These newly generated approximations, produced in an unsupervised manner, are then utilized to inform the training process in subsequent networks and subdomains. The gating mechanism plays a critical role in this process by determining the accuracy of the generated information.

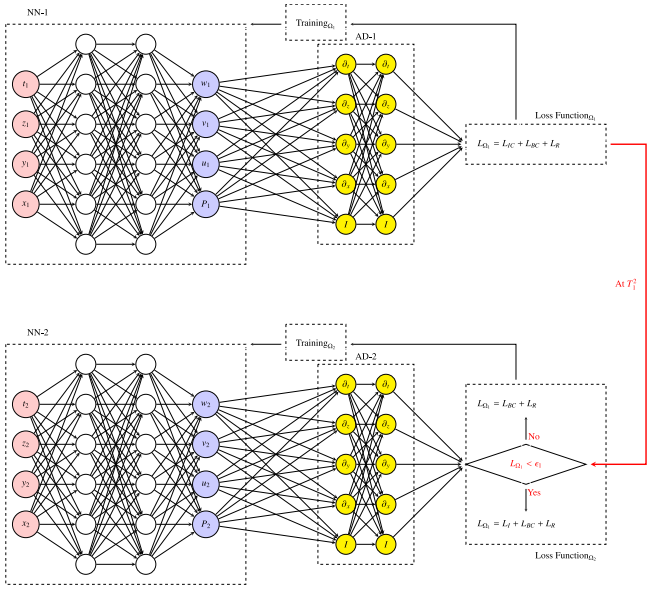

Fig. 3. Schematics of the DG-PINN. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

It does so by evaluating the error of the approximation within the transmitter model. Based on this error, the gating mechanism decides whether the generated data is sufficiently accurate to be transferred to subsequent subdomains. This iterative generation and validation of new approximations allow the model to continuously refine its solution over time, even in areas where initial data are sparse or unavailable. It is important to note that the newly generated and transferred information is treated as additional training data that was not originally available. At each time step, except for the very first one, the training data originally consists only of boundary conditions (supervised learning) and the residuals of the governing equations (unsupervised learning). The newly generated data, which the model creates as part of the iterative process, is incorporated as supplementary information to refine and enhance the training process at subsequent time steps.

The availability of sufficient boundary conditions for each subdomain is essential to avoid an under-determined system. When initial conditions are only provided to the first subdomain, a lack of adequate BC in subsequent subdomains can leave the system under-determined, thereby hindering the training process. In an extreme case, one might encounter a system where no BC or IC are available, relying solely on the minimization of the residuals of the governing equations. To mitigate this issue, a gating mechanism is typically employed to transfer additional information between subdomains. If adequate BC are unavailable, it may be necessary to relax the gating mechanism's transfer threshold, even if the transferred information is not perfectly accurate. This strategy ensures that subsequent subdomains do not remain under-determined, preserving the effectiveness of the training process across the domain. The optimal value of the transfer threshold, however, remains a hyperparameter that requires manual adjustment.

Clarification on the proposed method vs. Standard transfer learning

While our method shares some conceptual similarities with transfer learning, it fundamentally differs in both methodology and application:

Decomposition Approach: We decompose the entire temporal domain into multiple overlapping subdomains, each handled by a separate neural network model with its own model parameters. This allows each model to specialize in a specific region, capturing local dynamics more effectively.

Parallel Training: These subdomain models are trained in parallel, not sequentially.

Overlapping Zones: The overlapping regions between subdomains facilitate the transfer of information and ensure continuity and consistency across the entire domain. These overlaps allow models to share approximations in the overlapping zones, stitching together a coherent global solution.

Generative Mode: Our models generate new approximations at temporal points where no training data were originally provided. This is crucial for time-dependent problems with sparse initial data, as it allows the models to autonomously create necessary solution approximations based on the governing equations.

Iterative Refinement: The process allows for continuous refinement of the solution over time. By generating and validating new approximations, the models iteratively improve the solution, even in regions lacking initial data.

Use of the H^1 Norm of Error: We utilize the H^1 norm of the error when transferring approximations between subdomains. By incorporating both the function values and their gradients, the H^1 norm ensures that the transferred approximations maintain consistency with the governing PDEs and adhere to the underlying physics.

These features distinguish our approach from standard transfer learning, which typically involves sequentially training a single model on a source task and then fine-tuning it on a target task.

Terminology clarification

While we refer to this mechanism as "generative model", we acknowledge that it differs from the concept of "generative learning". Traditional generative learning models aim to model data distributions and generate new samples from these distributions. In our approach, the term "generative" is used to describe the process by which the model creates new approximations at temporal points or regions where no original training data is available. These approximations are deterministic outputs derived from solving the governing equations and are not probabilistically generated. The mechanism is "generative" in the sense that it autonomously propagates solutions to unexplored regions of the domain, enabling systematic information transfer and continuity across subdomains.

2.4.1. Analysis of the solution regularity

Using the H^1 function space instead of the L^2 in the loss function promotes solutions with higher regularity. This is because the H^1 norm penalizes not only the differences in function values but also the differences in their gradients, which encourages smoother solutions. It is important to note that this regularity assumption holds only when the approximations are sufficiently smooth, i.e., at least C^1 continuous. In cases where the functions are only C^0 continuous, the regularity argument breaks down since gradients may not be well-defined. Additionally, the regularity of the approximation is influenced by the activation function used in the network, as certain activation functions may limit the smoothness of the solutions. Let us formalize this using mathematical formulations and proofs:

H^1 **Norm**: The H^1 norm of a function $f(x)$ over a domain Ω is defined as:

$$\|f\|^2_{H^1(\Omega)} = \int_{\Omega} \left(|f(x)|^2 + |\nabla f(x)|^2 \right) dx$$

where $|\nabla f(x)|$ denotes the gradient of $f(x)$ and $|\cdot|$ denotes the L^2 norm.

Regularizing Effect: Adding the term $\|\nabla f(x)\|^2$ in the H^1 norm penalizes large gradients in the function $f(x)$. Minimizing this term encourages smoother solutions because abrupt changes or sharp variations in the function are penalized. However, the regularizing effect also depends on the smoothness of the approximation, which is

influenced by the choice of activation function. For instance, ReLU activations only guarantee C^0 continuity, limiting the smoothness of the approximation. On the other hand, the tanh activation function, which is a smooth, C^∞ continuous function, naturally provides a higher level of smoothness. By using tanh, the network approximations will exhibit smoother transitions, which aligns better with the assumptions of the H^1 norm. The proof of the regularizing effect in the model approximation is given in Appendix A.

Enhancement of the PINNs solution process using the H^1 norm of errors: Using the H^1 norm in the mean squared error of the loss function introduces additional constraints that further limit the space of admissible solutions. By considering both the function values and their gradients, the H^1 norm effectively narrows the solution space to functions that satisfy not only the governing equations but also possess the desired smoothness properties. This added constraint enhances the approximation by steering the optimization toward solutions that are physically and mathematically more consistent with the problem's nature.

- Alignment with Variational Principles: Many PDEs arise from variational formulations where the solution minimizes an energy functional in an H^1 space. By incorporating the H^1 norm into the loss function, we ensure that the PINN model adheres to the same minimization principles as the true solution. This alignment can potentially improve convergence to the true solution and enhance the accuracy of the approximation.
- Improved Optimization Landscape: Including derivative terms in the loss function can smooth the optimization landscape, making it easier for gradient-based optimization algorithms to find the global minimum. The presence of gradient information reduces the likelihood of encountering sharp minima or saddle points, leading to more stable convergence and reduced sensitivity to initialization and hyperparameters.
- Complementary Role to Tolerance and Method Order: While tolerance and the order of the numerical method are crucial factors dictating accuracy, the norm used in the loss function directly impacts how errors are measured and minimized during training. The H^1 norm provides a more comprehensive measure of error by accounting for both the solution and its derivatives. This can lead to more accurate and stable solutions, complementing the effects of tolerance and method order. By influencing the optimization process at each iteration, the H^1 norm guides the network toward better approximations that satisfy both the PDE and the desired regularity.

2.4.2. Gating mechanism

Since subdomain models are trained in parallel and separately, this dynamic generation and propagation of information may negatively impact the rate of convergence. Transferred data in the receiver model improves the training only when the generated data in the transmitter model (preceding subdomain) is accurate to a certain level. To address this requirement, we introduce a gating mechanism in the DG-PINN, such that data transmission between models is switched on only when the approximation in the transmitter subdomain reaches a predefined accuracy hyperparameter ϵ (Refer to the decision point in Fig. 3). Let ϵ be the predefined accuracy hyperparameter representing the threshold that the approximation in the transmitter subdomain needs to reach for data transmission to be enabled. Let $\mathcal{T}$ represent the transmitter subdomain model, and $\mathcal{R}$ represent the receiver subdomain model. We can introduce a binary gating variable γ that controls whether data transmission between the transmitter and receiver models is allowed. This gating variable γ takes the value 1 when data transmission is allowed and 0 otherwise. Mathematically, the gating mechanism can be formulated as follows:

$$\gamma = \begin{cases} 1, & \text{if accuracy}(\mathcal{T}) \geq \epsilon \\ 0, & \text{otherwise} \end{cases}$$

Here, accuracy($\mathcal{T}$) represents a measure of accuracy of the approximation in the transmitter subdomain, which could be, for example, the loss function value or the error in the solution. Finally, to mitigate excessive memory usage, as suggested, the gating mechanism can be implemented to operate at predetermined intervals, such as every 50 iterations.

2.4.3. Training

For the training of models in the DG-PINN, a distributed mini-batch gradient descent approach was employed. In this setup, both the training data and the model are discretized and processed separately and in parallel (refer to Fig. 3). Consequently, the training occurs independently for each subdomain, executed in parallel. The parameters of each model are updated utilizing the mini-batch gradient descent method. Considering different learning rates $\alpha^{(i)}$ for each parallel model $i = 1, 2, \ldots, N$, and using the loss function denoted by $L^{(i)}(\theta^{(i)})$, the update rule for each model's parameters $\theta^{(i)}$ in mini-batch gradient descent can be expressed as follows, such that the gradient calculation for ith model using a mini-batch of size m is

$$\nabla L^{(i)}(\theta^{(i)}) = \frac{1}{m}\sum_{j=1}^{m} \nabla L_j^{(i)}(\theta^{(i)}) \tag{28}$$

where $\nabla L_j^{(i)}(\theta^{(i)})$ is the gradient contribution from the jth data point in the mini-batch for the ith model. The update rule for ith model's parameter $\theta^{(i)}$ using the computed gradient and the corresponding learning rate is written as

$$\theta^{(i+1)} := \theta^{(i)} - \alpha^{(i)} \nabla L^{(i)}(\theta^{(i)}) \tag{29}$$

$$\theta^{(i+1)} := \theta^{(i)} - \frac{\alpha^{(i)}}{m}\sum_{j=1}^{m} \nabla L_j^{(i)}(\theta^{(i)}). \tag{30}$$

Accordingly, each parallel model i utilizes distinct learning rate $\alpha^{(i)}$ and updates parameters $\theta^{(i)}$ independently based on the gradients computed from its mini-batches.

The inputs of the NN are normalized to fall within the range of $[-1, 1]$ for each dimension. Similarly, the governing equations linked to these inputs are normalized using the same scaling factor. Consider three sets of data, including IC_i, BC_i, and R_i, where the subscript i denotes the ith data set related to the IC, BC, and the residuals of governing equations in the ith model

$$X_{\mathrm{IC}_i} = \{(x_{0i}, y_{0i}, z_{0i}, t_{0i}, u_{0i}, v_{0i}, w_{0i})\} \tag{31}$$

$$X_{\mathrm{BC}_i} = \{(x_{bi}, y_{bi}, z_{bi}, t_{bi}, u_{bi}, v_{bi}, w_{bi})\} \tag{32}$$

$$X_{\mathrm{R}_i} = \{(x_i, y_i, z_i, t_i)\}. \tag{33}$$

Splitting each data set i into mini-batches, N/m min-batches are created, each containing m samples. The kth mini-batch, denoted as B_k, consists of data indexed from $(k-1)\times m+1$ to $k\times m$. This can be mathematically represented as

$$B^k_{\mathrm{IC}_i} = \left\{(x_{0i}, y_{0i}, z_{0i}, t_{0i}, u_{0i}, v_{0i}, w_{0i})\right\}_{(k-1)\times m+1}^{k\times m} \tag{34}$$

$$B^k_{\mathrm{BC}_i} = \left\{(x_{bi}, y_{bi}, z_{bi}, t_{bi}, u_{bi}, v_{bi}, w_{bi})\right\}_{(k-1)\times m+1}^{k\times m} \tag{35}$$

$$B^k_{\mathrm{R}_i} = \left\{(x_i, y_i, z_i, t_i)\right\}_{(k-1)\times m+1}^{k\times m} \tag{36}$$

Permuting the data within mini-batches involves shuffling the indices within each mini-batch to randomize the order in which the data

Algorithm 1: Algorithm of the DG-PINN

Consider discretization of the temporal domain into N subdomains, where each subdomain has a separate model with trainable parameter θ_i and a loss function $L(\theta_i)$ such that

Initialization: Set initial parameters θ_i using [48];

for $n = 1$ **to** e **do**

  for $j = 1$ **to** s **do**

    for $i = 1$ **to** N **do**

      $L(\theta_i) = L_{\mathrm{IC}_i}(X_{\mathrm{IC}_i}) + L_{\mathrm{BC}_i}(X_{\mathrm{BC}_i}) + L_{\mathrm{R}_i}(X_{\mathrm{R}_i})$.

      $X_{\mathrm{IC}_i} = \{(x_{0i}, y_{0i}, z_{0i}, t_{0i}, u_{0i}, v_{0i}, w_{0i})\}, X_{\mathrm{BC}_i} = \{(x_{bi}, y_{bi}, z_{bi}, t_{bi}, u_{bi}, v_{bi}, w_{bi})\}, X_{\mathrm{R}_i} = \{(x_i, y_i, z_i, t_i)\}$

      $B^k_{\mathrm{IC}_i} = \left\{(x_{0i}, y_{0i}, z_{0i}, t_{0i}, u_{0i}, v_{0i}, w_{0i})\right\}^{k\times m}_{(k-1)\times m+1}$, $B^k_{\mathrm{BC}_i} = \left\{(x_{bi}, y_{bi}, z_{bi}, t_{bi}, u_{bi}, v_{bi}, w_{bi})\right\}^{k\times m}_{(k-1)\times m+1}$

      $B^k_{\mathrm{R}_i} = \left\{(x_i, y_i, z_i, t_i)\right\}^{k\times m}_{(k-1)\times m+1}$, $Permutation(k) = [perm(1), perm(2), ..., perm(m)]$

      $\theta = \theta - \alpha_i \cdot \nabla L(\theta)$

      if $j \bmod 50 = 0$ **then**

        if $L_{\theta_i} \leq \epsilon$ **then**

          $B^k_{\mathrm{IC}_{i+1}} = \{(x_0, y_0, z_0, t_{0_{int}}, u_{\theta_i}(0, t_{int}), v_{\theta_i}(0, t_{int}), w_{\theta_i}(0, t_{int}))\}$

        end

        else

          $L(\theta_{i+1}) = L_{\mathrm{BC}_{i+1}} + L_{\mathrm{R}_{i+1}}$

        end

      end

    end

  end

end

samples are fed into the learning algorithm. Given a mini-batch k with indices initially ordered as $[1, 2, \ldots, m]$, here is a mathematical representation for applying permutations within a mini-batch; let permutation (k) denote the permutation applied to the indices of mini-batch k. After applying a permutation to the kth mini-batch, the new order of indices becomes:

$$Permutation(k) = [perm(1), perm(2), \ldots, perm(m)] \tag{37}$$

where $perm(i)$ represents the ith element after permutation within mini-batch k. This operation randomizes the order of data samples within each mini-batch, contributing to a more effective training process in machine learning algorithms, especially during mini-batch gradient descent.

The DG-PINN is illustrated through Algorithm 1. N, e, and s denote the number of subdomains, epochs, and iterations, respectively. Additionally, α_i is the learning rate for each model and ϵ represents the predefined accuracy threshold for the gating mechanism. The training is based on the distributed mini-batch gradient descent approach, where various network parameters and learning rates are assigned to subdomains and models are trained in parallel. Notably, to mitigate excessive memory usage, the gating mechanism is activated at intervals of every 50 iteration, a configuration fine-tuned based on empirical observations gleaned from experimental analysis.

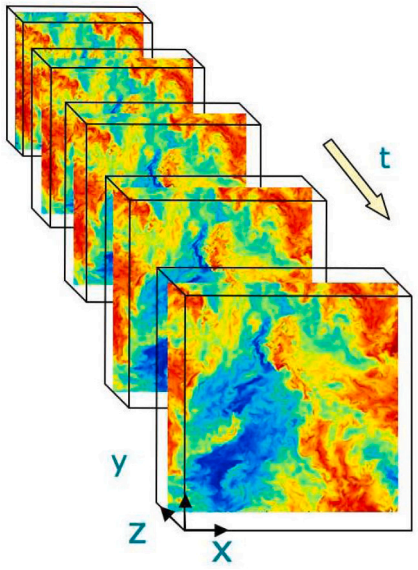

Fig. 4. Schematics of the problem domain [49–51] from https://turbulence.pha.jhu.edu.

3. Results and discussion

In this section, the objective is to evaluate how our proposed method, the DG-PINN, performs against the baseline PINNs and against traditional domain decomposition PINNs. The tangent hyperbolic (Tanh) function serves as the activation function, and the Glorot normal initialization method [48] is utilized for initializing all networks. As outlined in Algorithm 1, all networks are trained using a mini-batch gradient descent ADAM optimizer [28], using piece-wise decay of learning rates for each model. The temporal domain $[0, T]$ is discretized into subdomains $[0 : t_1], [t_1 : t_2], \ldots [t_{n-2} : t_{n-1}], [t_{n-1} : T]$. Controlling over the proper transfer of generated data from the previous subdomain to serve as the initial condition in the next subdomain, during the overlapping time instances $t_1, t_2, \ldots, t_{n-1}$, is managed by a gating mechanism. Implementation of automatic differentiation and construction of computational graphs are facilitated using TensorFlow [52].

The validation is performed using multiple cross-validations utilizing randomly generated and entirely unseen evaluation points distinct

from those employed during training. This process is followed by averaging the results. The number of test points corresponds to 25% of the total number of training points. In this section, the term PINNs refers to the baseline PINNs model, while DD-PINN represents a domain decomposition PINNs without overlapping constraints, and DG-PINN denotes our proposed method. Additionally, comparisons will be made between the DG-PINN and a counterpart of the DG-PINN model but using the L^2 norm of errors, to highlight the contribution os using H^1 norm of errors when propagating information between models. It is important to note that, for a fair comparison, the training time was kept the same across all competing methods in every comparison conducted in this study.

3.1. Problem setup

The test problem involves a channel flow with $Re = 9.994 \times 10^2$. This Reynolds number is almost the critical Reynolds number for the transition from laminar to turbulent flow in channel flows. According to several authoritative sources, turbulence can occur at this Reynolds number in channel flows [53,54]. We acknowledge that $Re \approx 1000$ is at the lower limit for turbulence in channel flows, but it is within the range where turbulence is expected to develop. Training and testing data for this study originate from the turbulent channel flow database from the Johns Hopkins Turbulence Database (JHTDB) available at https://turbulence.pha.jhu.edu/. These datasets are widely used in turbulence research and demonstrate turbulent characteristics at this Reynolds number. The spatial domain is $[0, 8\pi] \times [-1, 1] \times [0, 3\pi]$ and the time step in the online database is 0.0065. The viscosity is $\nu = 5 \times 10^{-5}$. The initial condition of the problem signifies a state of fully developed turbulence, as supported by the turbulence statistics provided in the reference datasets [51,55]. We note that detailed turbulence statistics from the reference data demonstrate the turbulent nature of the flow at this Reynolds number. These include:

- Mean Velocity Profiles: Exhibiting a clear logarithmic layer near the wall, which is characteristic of turbulent boundary layers in wall-bounded flows. The velocity profiles align with the classical law of the wall [55]:
$$U^+ = \frac{1}{k} ln y^+ + B, \tag{38}$$
where U^+ is the dimensionless velocity, y^+ is the dimensionless wall distance, k is the von Kármán constant, and B is an empirical constant.
- Reynolds Stress Profiles: Showing significant turbulent shear stresses and normal stresses, indicating turbulent momentum transfer by turbulent eddies [55]. The Reynolds shear stress $-\overline{u'v'}$ has substantial values across the channel height, with peaks away from the wall.
- Turbulent Kinetic Energy (TKE) Profiles: Displaying high levels of TKE near the wall, characteristic of wall-bounded turbulent flows [56]. The TKE decreases towards the channel centerline, reflecting the distribution of energetic turbulent motions.
- Energy Spectra: Demonstrating an inertial subrange with a $-5/3$ slope in the spectral energy density when plotted on log–log scales, consistent with Kolmogorov's theory of turbulence [57]. The spectra show energy distributed across a wide range of scales, from large to small eddies.

While these key turbulence statistics illustrate the turbulent nature of the flow at this Reynolds number, a detailed analysis is beyond the scope of this work. For in-depth discussions on turbulence characteristics, we refer interested readers to the cited references.

The architecture of all the networks in this study is based on the FFNN architecture. Network input and output are four-dimensional, where (x, y, z, t) serve as inputs and (u, v, w, P) are outputs. In case studies I and II, we have considered two different domains with different spatial and temporal sizes at various locations within the channel.

Table 2
Settings of the training in case study I.

# Epochs	# Iterations	Learning rate
300	150	1×10^{-3}
200	150	2×10^{-4}
200	150	4×10^{-5}
100	150	9×10^{-6}
50	150	9×10^{-7}

Table 3
Networks settings and memory usage in case study I.

Method	Architecture	GPU use (Gb)
Baseline PINNs	$(1) \times 10 \times 300$	8.6
DD-PINN	$(10) \times 10 \times 150$	2.5
DG-PINN	$(10) \times 10 \times 150$	2.5

It is important to note that the quantitative results presented in this study are the average outcome obtained from 10 consecutive tests. Schematics of the problem domain are shown in Fig. 4. The data and the codes utilized in this manuscript are publicly accessible on GitHub at https://github.com/AmirhosseinnnKhademi/DG-PINN.git.

3.2. Case study I

To study the performance of the DG-PINN, a domain is considered with dimensions $[12.47, 12.66] \times [-0.90, -0.70] \times [4.61, 4.82]$ for x, y, and z respectively. The temporal domain consists of 110 time steps. According to the description of the database [58], there is no viscous sublayer within the boundaries of this simulation. There are 10,000 collocation points and 6644 BC points within each time step. Within the first time step, there are 33,524 initial condition (IC) points. The adjustments for the piece-wise decay of learning rates, number of epochs, and number of iterations are specified in Table 2. The architectural configuration for networks in case study I is outlined in Table 3. The second column shows the architecture of the networks, where the number in parentheses represents the number of subdomains, followed by the number of hidden layers and the third is the number of neurons in each hidden layer. Additionally, memory usage is given for each model. The comparison between the DD-PINN and the DG-PINN is intended to highlight the significance of the generative model. This table highlights the efficacy of domain discretization, enabling the implementation of a distributed training strategy that results in more efficient memory usage. The instantaneous velocity fields in the $y - z$ plane at $x = 12.66$ approximated by the DG-PINN are compared with those predicted by the baseline PINNs and the reference DNS solution in Figs. 5, 6, and 7 at time instances t_8, t_{65}, t_{95}, respectively. In this simulation, both the DG-PINN and the DD-PINN decompose the temporal domain into 10. As it is observable, the DG-PINN can simulate the 3D turbulent fluid flow with high accuracy and successfully capture the details. On the other hand, capturing general patterns while struggling with finer details was observed in the DD-PINN. Finally, the baseline PINNs could not converge to an accurate approximation within this limited number of epochs. Previous research by Jin et al. [25] indicated that significantly more epochs and iterations were required for the PINNs to converge in a similar problem.

To be more detailed, quantitative results comparing the errors among PINNs, DD-PINNs, and the DG-PINN are presented in Fig. 8. While this figure confirms the findings of Figs. 5 to 7, they also reveal an additional point; as time progressed away from the initial time step with well-defined IC, the accuracy of the prediction made by the baseline PINNs drastically decreases. Although the DD-PINN is able to predict more accurate approximations in the subsequent time steps, the accuracy is not comparable with the initial time steps due to the lack

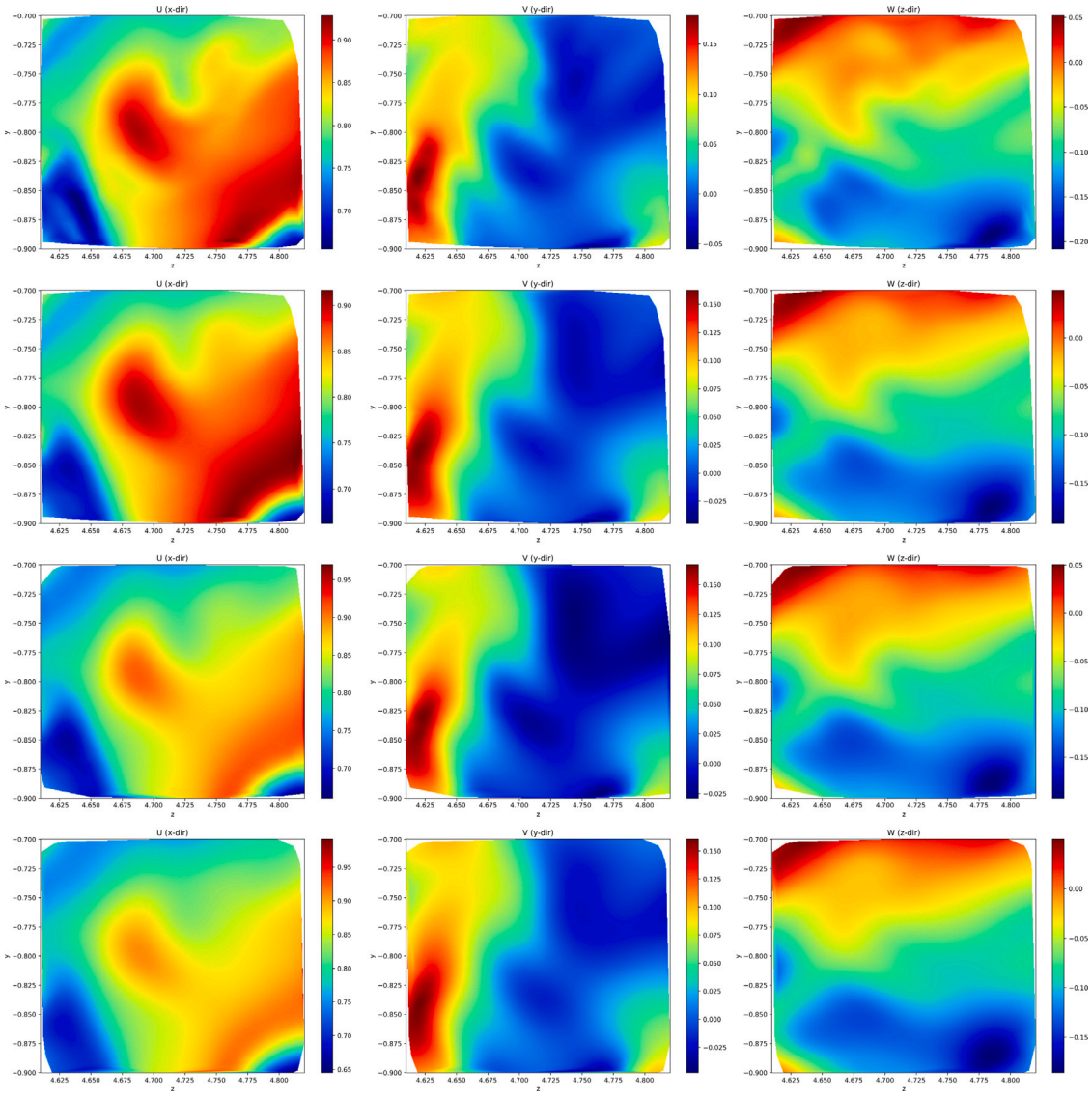

Fig. 5. Instantaneous velocity fields at $t_8 = 8 \times 0.0065$: From top to bottom: DNS solution, DG-PINN, DD-PINN, and Baseline PINNs.

of IC. However, the DG-PINN is able to maintain the approximation accuracy at time instances where the model lacks IC, thanks to the discretized generative model. In particular, in the y and z directions where the PINNs' prediction error increases by approximately 25 percent, the DG-PINN significantly improves the accuracy at distant time steps.

The findings suggest that the wall-normal and spanwise velocities exhibit significantly greater relative L^2 errors compared to the streamwise velocity. This discrepancy arises due to the streamwise velocity's substantially larger amplitude in comparison to the other two velocity components. Lastly, in Fig. 9, the approximation of instantaneous pressure fields by the DG-PINN is compared against the DNS solution. The results refer to the $y - z$ plane at $x = 12.55$ at time step $t_{15} = 15 \times 0.0065$. The DG-PINN demonstrates an acceptable agreement with the reference solution. It is intriguing to observe that, despite no explicit training data for pressure being provided to the model, it successfully learns and approximates the pressure field solely through unsupervised physics-informed learning. As illustrated by Raissi et al. [10], the observed difference in magnitude between the exact and predicted pressure is justified by the inherent characteristics of the incompressible Navier–Stokes system. In this system, the pressure field is identifiable only up to a constant. To underscore the significance of employing the H^1 function space, over the L^2 as used in previous discretized PINNs models, a comparison between the DG-PINN and its counterpart model utilizing the L^2 norm is shown in Fig. 10. To simplify visualization and maintain clarity despite the inherent complexity of the 3D time-dependent problem, the plots in the figure depict solution approximations and their derivatives along the y direction. Six time steps are assigned to each subdomain and there is one overlapping time step between each pair of subdomains. Other settings and adjustments remain same as before. The subplots in the left column depict results for the model using the L^2 norm, while those in the right column represent the results for the DG-PINN utilizing the H^1 norm. In the left column of the figure, it is observed that in a limited number of training epochs, the discretized model utilizing L^2 norm exhibits differing gradient values between adjacent subdomains, suggesting the absence of a well-defined gradient for the global solution. Moreover, the model faces challenges in fully preserving continuity. Conversely, in the right column, representing the DG-PINN model employing the H^1 function space, despite the presence of sharp edges, the H^1 function space ensures consistent first derivatives. Solutions obtained through

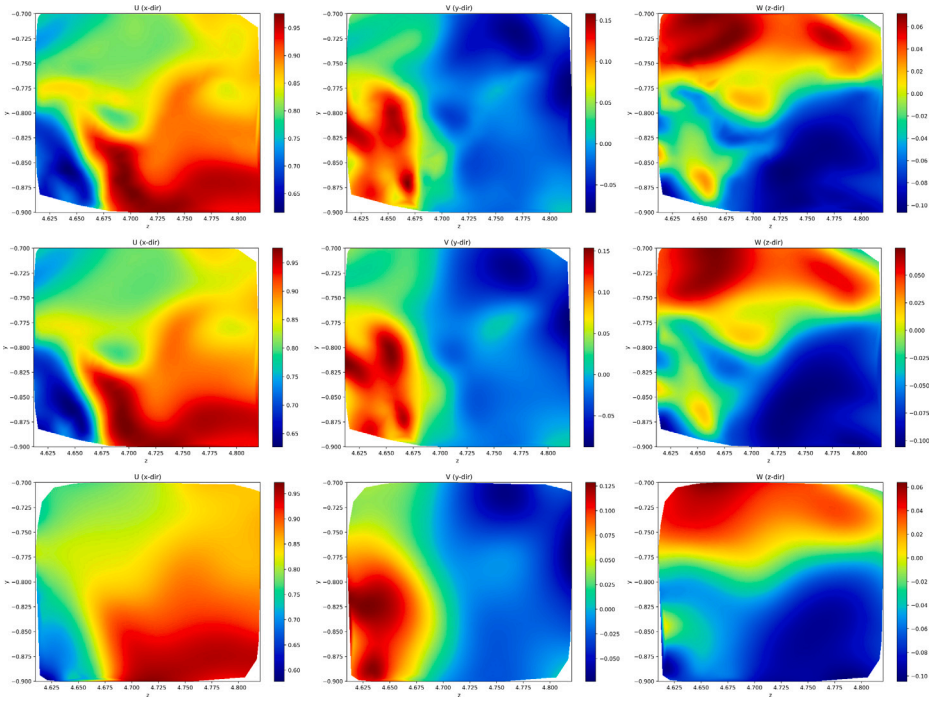

Fig. 6. Instantaneous velocity fields at $t_{65} = 65 \times 0.0065$: From top to bottom: DNS solution, DG-PINN, and Baseline PINNs.

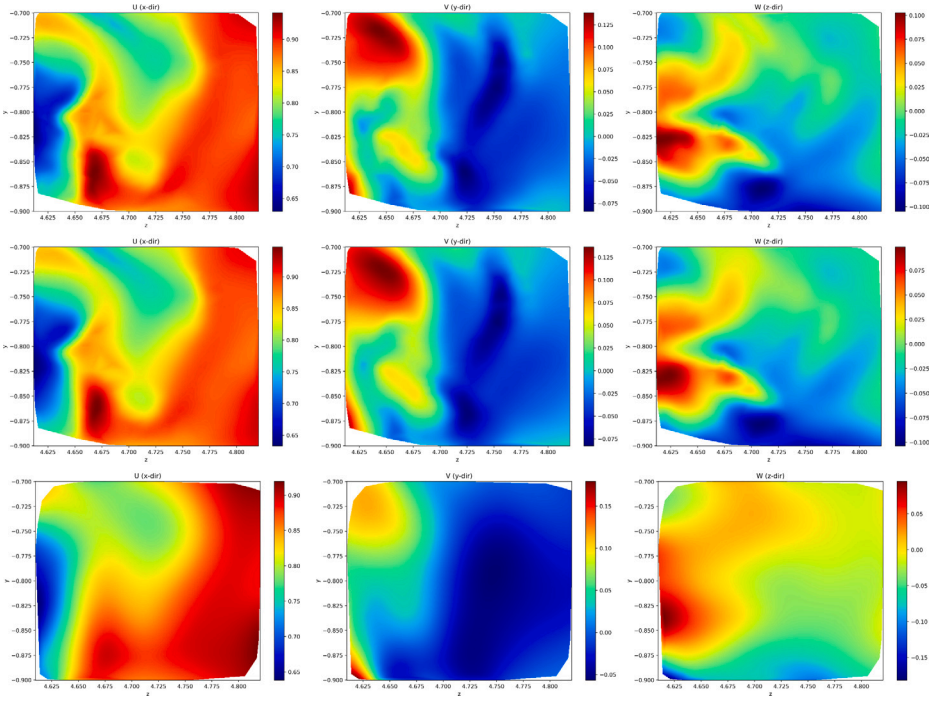

Fig. 7. Instantaneous velocity fields at $t_{95} = 95 \times 0.0065$: From top to bottom: DNS solution, DG-PINN, and Baseline PINNs.

the DG-PINN model demonstrate continuity, as the approximations by different subdomains at the boundary remain consistent. This implies the existence of continuity and first derivatives in the global solutions, a feature lacking in previous domain decomposition techniques in PINNs. For further insights into subsequent time steps and subdomains, additional plots, all confirming the conclusion here, are provided in Figs. 12 and 13 in the Appendix.

3.3. Case study II

To reflect the ability of the DG-PINN to simulate fluid flows with more complicated phenomena, a problem involving a larger domain, $[12.25, 12.75] \times [-1.0, -0.5] \times [4.5, 5.0]$, is considered. Within this larger domain, various interactions between different scales of eddies happen, which covers the range involving the viscous sub-layer, the log-law region, the law of the wall, the buffer layer, and the outer layer [51, 58]. The non-dimensional time domain is limited to 16 time steps,

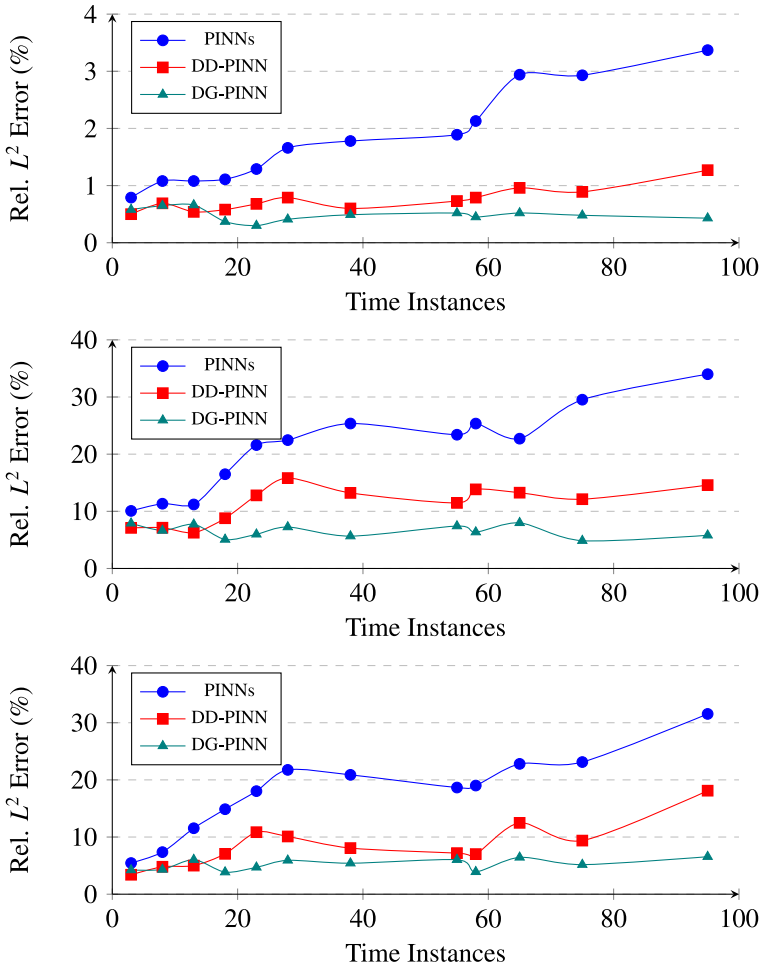

Fig. 8. Comparison of the error: Top: x direction, middle: y direction, and bottom: z direction.

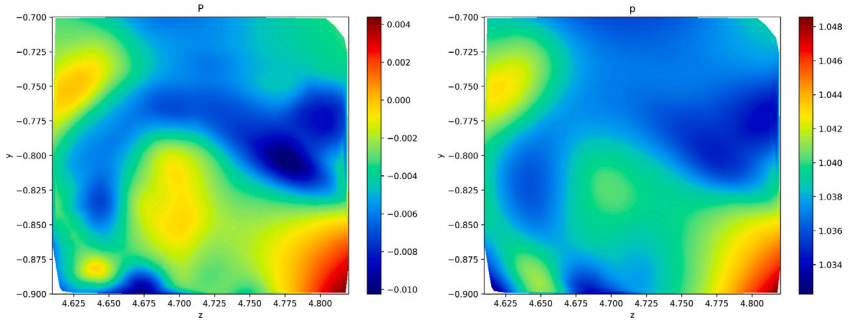

Fig. 9. Pressure fields at $t_{15} = 15.0065$. left: DNS solution; right: DG-PINN approximation.

where each time interval is 0.0065. There are 50,000 collocation points inside the domain and 26,048 boundary points at each time step, with 147,968 points corresponding to the IC at the initial time step. Finally, there have been 1400 epochs in the training process, with the details presented in Table 4. In this simulation, the temporal domain is discretized into 4 subdomains. Each subdomain covers 4 consecutive time steps and shares one overlapping time step with the adjacent subdomain, resulting in a total of 16 non-repeated time steps. The computational graph adopts an FFNN architecture with 9 hidden layers, each containing 300 neurons. The input and output layers have dimensions of 4.

Fig. 11 compares the approximations of the instantaneous velocity and pressure fields by the DNS, and the DG-PINN at $t_{13} = 13 \times 0.0065$ in the $y - z$ plane at $x = 12.66$. As proposed by Fig. 11, the DG-PINN demonstrates the capability to accurately approximate solutions for the turbulent 3D Navier–Stokes equations in a large domain, without employing any turbulence filter, directly approximating the Navier–Stokes equations. Table 5 shows the relative L^2 error in approximations

Table 4
Settings of the training in case study II.

# Epochs	# Iterations	Learning rate
50	150	1×10^{-3}
500	150	5×10^{-4}
400	150	7×10^{-5}
400	150	1×10^{-5}
50	150	5×10^{-6}

a: Continuity of u in overlapping zone (y direction)

b: Continuity of v in overlapping zone (y direction)

c: Continuity of w in overlapping zone (y direction)

d: First derivative of u in overlapping zone (y direction)

e: First derivative of v in overlapping zone (y direction)

f: First derivative of w in overlapping zone (y direction)

Fig. 10. Solution regularity analysis at 5th time step. Right column: DG-PINN approximations using H^1 norm; left column: a counterpart model using L^2 norm resembling traditional decomposition methods.

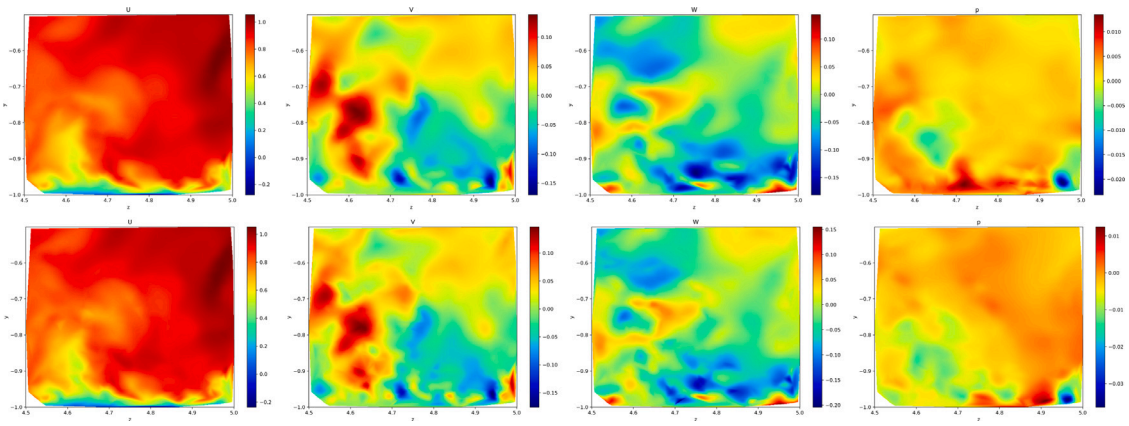

Fig. 11. Velocity and pressure fields at $t_{13} = 13 \times 0.0065$. Top: DNS solutions; bottom: DG-PINN approximations.

Table 5
DG-PINN velocity approximation accuracy in case study II.

Time step	L^2 error x direction	L^2 error y direction	L^2 error z direction
0	1.4%	12.2%	10.1%
4	2.2%	16.3%	13.7%
8	1.7%	15.1%	12.4%
12	2.4%	17.9%	14.1%

by the DG-PINN in various sample time steps. Sustained accuracy of approximations over time, particularly in the absence of IC, even within an expanded domain, highlights the effectiveness of this novel technique. This is worth mentioning that the baseline PINNs fail to converge to an accurate approximation within 1400 training epochs for such a large domain with noticeably more training and collocation points.

4. Conclusion

This research introduced a novel methodology within Physics-Informed Neural Networks known as the discretized generative model PINNs. The DG-PINN is developed through discretization of the temporal domain, employing multiple models and generation of new training data, incorporating a gating mechanism for optimal regulation of data transfer among models, formulation of extra loss terms within H^1 function space, and using a distributed training strategy. This methodology was applied to a time-dependent 3D turbulent channel flow at $Re \approx 1000$, governed by the incompressible Navier–Stokes equations, and achieved good agreements with the reference solution. In the first case study, a comparison was made between the instantaneous velocity fields generated by the baseline PINNs and those of the DG-PINN over an extended period. The DG-PINN exhibits superior accuracy compared to the baseline PINNs while utilizing less memory thanks to a distributed training strategy made possible by model discretization. The approximation error of the baseline PINNs increases significantly over time, particularly in subsequent time steps where initial conditions are not available. In contrast, the DG-PINN successfully maintains accuracy throughout. Furthermore, quantitative results demonstrated that employing H^1 function space enhanced the regularity of the global solutions in comparison to previous domain decomposition PINNs techniques utilizing L^2 function space. Interestingly, the DG-PINN accurately approximated pressure field patterns without utilizing any pressure training data within the complex 3D time-dependent turbulent flow regime. In the second case study, covering a more extensive spatial domain, which includes the viscous sub-layer, log-law region, law of the wall, buffer layer, and outer layer, the DG-PINN converges to a relatively accurate approximation, whereas the baseline PINNs fails to converge within the specified number of training epochs. The DG-PINN allows for the simultaneous use of multiple models, paving the way for future research to deploy our method with different turbulent models, thereby enhancing accuracy in broader domains characterized by diverse flow regimes.

CRediT authorship contribution statement

Amirhossein Khademi: Writing – original draft, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization, Data curation, Project administration, Resources, Visualization, Writing – review & editing. **Erfan Salari:** Writing – review & editing, Investigation. **Steven Dufour:** Writing – review & editing, Supervision, Project administration.

Declaration of competing interest

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

Appendix A

A.1. Methodology: Solution regularity analysis

Proof. Consider a sufficiently smooth function $f(x)$ and its approximate solution $g(x)$ obtained through optimization. We aim to show that minimizing the H^1 norm of $g(x)$ encourages smoothness. Specifically, this proof assumes that $f(x)$ and $g(x)$ are at least C^1 continuous.

Let $E(f,g) = \frac{1}{2}\|f-g\|^2_{H^1(\Omega)}$ denote the error between the true function $f(x)$ and the approximate solution $g(x)$ in the H^1 norm.

We can expand $E(f,g)$ as follows:

$$E(f,g) = \frac{1}{2}\left(\|f-g\|^2_{L^2} + \|\nabla(f-g)\|^2_{L^2}\right)$$

Now, let us analyze the second term $\|\nabla(f-g)\|^2_{L^2}$. By the triangle inequality, we have:

$$\|\nabla(f-g)\|^2_{L^2} \leq \|\nabla f\|^2_{L^2} + \|\nabla g\|^2_{L^2}$$

Since the true function $f(x)$ is assumed to be smooth (i.e., at least C^1 continuous), $\|\nabla f\|^2_{L^2}$ is bounded.

Minimizing $E(f,g)$ with respect to $g(x)$ entails minimizing both $\|f-g\|^2_{L^2}$ and $\|\nabla(f-g)\|^2_{L^2}$. Minimizing the latter term $\|\nabla(f-g)\|^2_{L^2}$ effectively encourages $g(x)$ to be smooth by penalizing sharp variations.

a: Continuity of u in overlapping zone (y direction)

b: Continuity of v in overlapping zone (y direction)

c: Continuity of w in overlapping zone (y direction)

d: First derivative of u in overlapping zone (y direction)

e: First derivative of v in overlapping zone (y direction)

f: First derivative of w in overlapping zone (y direction)

Fig. 12. Solution regularity analysis at 10th time step: DG-PINN using H^1 norm (right) vs. a counterpart model using L^2 norm resembling traditional decomposition methods (left).

a: Continuity of u in overlapping zone (y direction)

b: Continuity of v in overlapping zone (y direction)

c: Continuity of w in overlapping zone (y direction)

d: First derivative of u in overlapping zone (y direction)

e: First derivative of v in overlapping zone (y direction)

f: First derivative of w in overlapping zone (y direction)

Fig. 13. Solution regularity analysis at 20th time step: DG-PINN using H^1 norm (right) vs. a counterpart model using L^2 norm resembling traditional decomposition methods (left).

However, the regularity of the approximation $g(x)$ is affected by the activation function used in the neural network. For instance, ReLU-based networks, which produce piecewise linear approximations, may not fully benefit from the H^1 norm's regularizing effect due to their limited smoothness. In contrast, networks using smoother activation functions, such as tanh or softplus, produce much smoother approximations, better aligning with the assumptions of the H^1 norm. Therefore, using the H^1 norm instead of the L^2 in the loss function promotes solutions with higher regularity, provided that the approximations are sufficiently smooth and the choice of activation function supports this regularity.

A.2. Results: Solution regularity analysis

See Figs. 12 and 13.

Appendix B. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.ijnonlinmec.2024.104988.

Data availability

The data and the codes used in this manuscript are publicly accessible on GitHub at https://github.com/AmirhosseinnnKhademi/DG-PINN.git.

References

[1] F. Zacchei, F. Rizzini, G. Gattere, A. Frangi, A. Manzoni, Neural networks based surrogate modeling for efficient uncertainty quantification and calibration of mems accelerometers, Int. J. Non-Linear Mech. 167 (2024) 104902.

[2] W. Jia, M. Hu, W. Zan, F. Ni, Data-driven methods for the inverse problem of suspension system excited by jump and diffusion stochastic track excitation, Int. J. Non-Linear Mech. 166 (2024) 104819.

[3] Y. Zhang, R.P. Dwight, M. Schmelzer, J.F. Gómez, Z.-h. Han, S. Hickel, Customized data-driven rans closures for bi-fidelity les–rans optimization, J. Comput. Phys. 432 (2021) 110153.

[4] Z. Gou, X. Tu, S.V. Lototsky, R. Ghanem, Switching diffusions for multiscale uncertainty quantification, Int. J. Non-Linear Mech. (2024) 104793.

[5] V. Mahesh, Artificial neural network (ann) based investigation on the static behaviour of piezo-magneto-thermo-elastic nanocomposite sandwich plate with cnt agglomeration and porosity, Int. J. Non-Linear Mech. 153 (2023) 104406.

[6] S.K. Mitusch, S.W. Funke, M. Kuchta, Hybrid fem-nn models: Combining artificial neural networks with the finite element method, J. Comput. Phys. 446 (2021) 110651.

[7] F.M. d. Lara, E. Ferrer, Accelerating high order discontinuous galerkin solvers using neural networks: 3d compressible navier-stokes equations, J. Comput. Phys. (2023) 112253.

[8] W. Chen, Q. Wang, J.S. Hesthaven, C. Zhang, Physics-informed machine learning for reduced-order modeling of nonlinear problems, J. Comput. Phys. 446 (2021) 110666.

[9] Y. Kim, Y. Choi, D. Widemann, T. Zohdi, A fast and accurate physics-informed neural network reduced order model with shallow masked autoencoder, J. Comput. Phys. 451 (2022) 110841.

[10] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.

[11] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, 2017, arXiv preprint arXiv:1711.10561.

[12] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, J. Marchine Learn. Res. 18 (2018) 1–43.

[13] Z. Mao, A.D. Jagtap, G.E. Karniadakis, Physics-informed neural networks for high-speed flows, Comput. Methods Appl. Mech. Engrg. 360 (2020) 112789.

[14] A.D. Jagtap, E. Kharazmi, G.E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems, Comput. Methods Appl. Mech. Engrg. 365 (2020) 113028.

[15] A.D. Jagtap, G.E. Karniadakis, Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, in: AAAI Spring Symposium: MLPS, 2021, pp. 2002–2041.

[16] K. Shukla, A.D. Jagtap, G.E. Karniadakis, Parallel physics-informed neural networks via domain decomposition, J. Comput. Phys. 447 (2021) 110683.

[17] E. Kharazmi, Z. Zhang, G.E. Karniadakis, Variational physics-informed neural networks for solving partial differential equations, 2019, arXiv preprint arXiv:1912.00873.

[18] E. Kharazmi, Z. Zhang, G.E. Karniadakis, Hp-vpinns: Variational physics-informed neural networks with domain decomposition, Comput. Methods Appl. Mech. Engrg. 374 (2021) 113547.

[19] A. Khademi, S. Dufour, A novel discretized physics-informed neural network model applied to the navier-stokes equations, Phys. Scr. (2024).

[20] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, SIAM J. Sci. Comput. 43 (2021) A3055–A3081.

[21] S. Wang, H. Wang, P. Perdikaris, Improved architectures and training algorithms for deep operator networks, J. Sci. Comput. 92 (2022) 35.

[22] L. McClenny, U. Braga-Neto, Self-adaptive physics-informed neural networks using a soft attention mechanism, 2020, arXiv preprint arXiv:2009.04544.

[23] O. Hennigh, S. Narasimhan, M.A. Nabian, A. Subramaniam, K. Tangsali, Z. Fang, M. Rietmann, W. Byeon, S. Choudhry, Nvidia simnet™: An ai-accelerated multi-physics simulation framework, in: International Conference on Computational Science, Springer, 2021, pp. 447–461.

[24] A. Kendall, Y. Gal, R. Cipolla, Multi-task learning using uncertainty to weigh losses for scene geometry and semantics, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 7482–7491.

[25] X. Jin, S. Cai, H. Li, G.E. Karniadakis, Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations, J. Comput. Phys. 426 (2021) 109951.

[26] L.D. McClenny, U.M. Braga-Neto, Self-adaptive physics-informed neural networks, J. Comput. Phys. 474 (2023) 111722.

[27] Y. Liu, W. Liu, X. Yan, S. Guo, C.-a. Zhang, Adaptive transfer learning for pinn, J. Comput. Phys. 490 (2023) 112291.

[28] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.

[29] S. Wang, S. Sankaran, P. Perdikaris, Respecting causality is all you need for training physics-informed neural networks, 2022, arXiv preprint arXiv:2203.07404.

[30] Y. Song, H. Wang, H. Yang, M.L. Taccari, X. Chen, Loss-attentional physics-informed neural networks, J. Comput. Phys. 501 (2024) 112781.

[31] M. Penwarden, A.D. Jagtap, S. Zhe, G.E. Karniadakis, R.M. Kirby, A unified scalable framework for causal sweeping strategies for physics-informed neural networks (pinns) and their temporal decompositions, J. Comput. Phys. 493 (2023) 112464.

[32] A.F. Psaros, K. Kawaguchi, G.E. Karniadakis, Meta-learning pinn loss functions, J. Comput. Phys. 458 (2022) 111121.

[33] M. Penwarden, S. Zhe, A. Narayan, R.M. Kirby, A metalearning approach for physics-informed neural networks (pinns): Application to parameterized pdes, J. Comput. Phys. 477 (2023) 111912.

[34] M. Raissi, G.E. Karniadakis, Hidden physics models: Machine learning of nonlinear partial differential equations, J. Comput. Phys. 357 (2018) 125–141.

[35] S.K. Biswas, N. Anand, Three-dimensional laminar flow using physics informed deep neural networks, Phys. Fluids 35 (2023).

[36] S. Hu, M. Liu, S. Zhang, S. Dong, R. Zheng, Physics-informed neural network combined with characteristic-based split for solving navier–stokes equations, Eng. Appl. Artif. Intell. 128 (2024) 107453.

[37] S. Hijazi, M. Freitag, N. Landwehr, Pod-galerkin reduced order models and physics-informed neural networks for solving inverse problems for the navier–stokes equations, Adv. Model. Simul. Eng. Sci. 10 (2023) 5.

[38] J. Cho, S. Nam, H. Yang, S.-B. Yun, Y. Hong, E. Park, Separable physics-informed neural networks, Adv. Neural Inf. Process. Syst. 36 (2024).

[39] S.J. Anagnostopoulos, J.D. Toscano, N. Stergiopulos, G.E. Karniadakis, Residual-based attention in physics-informed neural networks, Comput. Methods Appl. Mech. Engrg. 421 (2024) 116805.

[40] G. Jin, J.C. Wong, A. Gupta, S. Li, Y.-S. Ong, Fourier warm start for physics-informed neural networks, Eng. Appl. Artif. Intell. 132 (2024) 107887.

[41] S. Wang, M. Nikfar, J.C. Agar, Y. Liu, Stacked deep learning models for fast approximations of steady-state navier–stokes equations for low re flow, Intell. Comput. 3 (2024) 0093.

[42] A. Beck, D. Flad, C.-D. Munz, Deep neural networks for data-driven les closure models, J. Comput. Phys. 398 (2019) 108910.

[43] Y. Zhao, H.D. Akolekar, J. Weatheritt, V. Michelassi, R.D. Sandberg, Rans turbulence model development using cfd-driven machine learning, J. Comput. Phys. 411 (2020) 109413.

[44] S. Hanrahan, M. Kozul, R. Sandberg, Studying turbulent flows with physics-informed neural networks and sparse data, Int. J. Heat Fluid Flow 104 (2023) 109232.

[45] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (1989) 359–366.

[46] D.S. Berman, A.L. Buczak, J.S. Chavis, C.L. Corbett, A survey of deep learning methods for cyber security, Information 10 (2019) 122.

[47] S. Cuomo, V.S.D. Cola, F. Giampaolo, G. Rozza, M. Raissi, F. Piccialli, Scientific machine learning through physics–informed neural networks: Where we are and what's next, J. Sci. Comput. 92 (2022) 88.
[48] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, in: JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.
[49] E. Perlman, R. Burns, Y. Li, C. Meneveau, Data exploration of turbulence simulations using a database cluster, in: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, 2007, pp. 1–11.
[50] Y. Li, E. Perlman, M. Wan, Y. Yang, C. Meneveau, R. Burns, S. Chen, A. Szalay, G. Eyink, A public turbulence database cluster and applications to study lagrangian evolution of velocity increments in turbulence, J. Turbul. (2008) N31.
[51] J. Graham, K. Kanov, X. Yang, M. Lee, N. Malaya, C. Lalescu, R. Burns, G. Eyink, A. Szalay, R. Moser, et al., A web services accessible database of turbulent channel flow and its use for testing a new integral wall model for les, J. Turbul. 17 (2016) 181–215.
[52] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., $\{TensorFlow\}$: a system for $\{Large-Scale\}$ machine learning, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265–283.
[53] H. Schlichting, K. Gersten, H. Schlichting, K. Gersten, Boundary-layer control (suction/blowing), Bound.-Layer Theory (2000) 291–320.
[54] F.M. White, J. Majdalani, Viscous Fluid Flow, vol. 3, McGraw-Hill New York, 2006.
[55] M. Lee, R.D. Moser, Direct numerical simulation of turbulent channel flow up to, J. Fluid Mech. 774 (2015) 395–415.
[56] S. Hoyas, J. Jiménez, Scaling of the velocity fluctuations in turbulent channels up to $re_\tau = 2003$, Phys. Fluids 18 (2006).
[57] J.C. Del Alamo, J. Jiménez, Spectra of the very large anisotropic scales in turbulent channels, Phys. Fluids 15 (2003) L41–L44.
[58] S. Cant, Sb Pope, Turbulent Flows, cambridge University Press, cambridge, uk, 2000, p. 771. Combust. Flame 125 (2001) 1361–1362.