Physics of Fluids

RESEARCH ARTICLE | FEBRUARY 28 2024

Physics-informed neural networks with domain decomposition for the incompressible Navier–Stokes equations

Linyan Gu (古林燕) ; Shanlin Qin (覃善林) ; Lei Xu (徐磊) ; Rongliang Chen (陈荣亮)

Physics of Fluids 36, 021914 (2024)
https://doi.org/10.1063/5.0188830

Physics of Fluids ARTICLE pubs.aip.org/aip/pof

Physics-informed neural networks with domain decomposition for the incompressible Navier–Stokes equations

Cite as: Phys. Fluids **36**, 021914 (2024); doi: 10.1063/5.0188830
Submitted: 24 November 2023 · Accepted: 31 January 2024 ·
Published Online: 28 February 2024

Linyan Gu (古林燕),[1] Shanlin Qin (覃善林),[2] Lei Xu (徐磊),[2,a)] and Rongliang Chen (陈荣亮)[2,3,a)]

AFFILIATIONS

[1]School of Mathematics, Sun Yat-Sen University, Guangzhou, Guangdong 510275, China
[2]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, Guangdong 518055, China
[3]Shenzhen Key Laboratory for Exascale Engineering and Scientific Computing, Shenzhen, Guangdong 518055, China

[a)]Authors to whom correspondence should be addressed: lei.xu@siat.ac.cn and rl.chen@siat.ac.cn

ABSTRACT

Physics-informed neural network (PINN) has emerged as a promising approach for solving differential equations in recent years. However, their application to large-scale complex problems has faced challenges regarding accuracy and efficiency. To address these limitations, domain decomposition has gained popularity as an effective strategy. This paper studies a domain decomposition PINN method for solving incompressible Navier–Stokes equations. We assess the method's predicted accuracy, convergence, and the impact of different strategies on performance. In the domain decomposition PINN method, individual PINN is employed for each subdomain to compute local solutions, which are seamlessly connected by enforcing additional continuity conditions at the interfaces. To improve the method's performance, we investigate various continuity conditions at the interfaces and analyze their influence on the predictive accuracy and interface continuity. Furthermore, we introduce two approaches: the dynamic weight method and a novel neural network architecture incorporating attention mechanisms, both aimed at mitigating gradient pathologies commonly encountered in PINN methods. To demonstrate the effectiveness of the proposed method, we apply it to a range of forward and inverse problems involving diverse incompressible Navier–Stokes flow scenarios. This includes solving benchmark problems such as the two-dimensional (2D) Kovasznay flow, the three-dimensional (3D) Beltrami flow, the 2D lid-driven cavity flow, and the 2D cylinder wake. Additionally, we conduct 3D blood flow simulations for synthetic flow geometries and real blood vessels. The experimental results demonstrate the capability and versatility of the domain decomposition PINN method in accurately solving incompressible Navier–Stokes flow problems.

I. INTRODUCTION

Over the past decade, the field of machine learning has experienced remarkable advancements, particularly in deep learning. This revolutionary approach has brought significant improvements to diverse disciplines, including natural language processing, computer vision, and healthcare, among others. In recent years, there has been growing interest in the concept of scientific machine learning, which involves the integration of machine learning techniques with scientific computing. One noteworthy technique in this field is the use of physics-informed neural networks (PINNs).[1] PINNs leverage the capabilities of neural networks to approximate solutions of differential equations by directly incorporating the governing equations into the network architecture. The foundations of PINNs can be traced back to earlier concepts proposed in the 1990s,[2,3] which have been further extended and refined using modern advancements in deep learning by Raissi *et al.*[1] An important advantage of PINNs is their mesh-free nature, which helps overcome the challenges associated with the curse of dimensionality. Furthermore, PINNs provide an elegant framework for addressing both forward and inverse problems through a unified optimization procedure.[1,4] This is made possible by the inherent tractability of automatic differentiation in contemporary deep learning frameworks. Additionally, PINNs excel at leveraging large volumes of multi-fidelity observation data, enabling them to extract meaningful features and integrate valuable physical insights. In summary, by combining the strengths of deep learning and scientific computing, PINNs present a promising approach for addressing complex problems across various scientific domains.

Physics-informed neural networks (PINNs) have revolutionized the field of computational fluid dynamics (CFD) with their versatile applications in solving a variety of partial differential equations (PDEs). These include integer-order, fractional, and stochastic PDEs.[5] Particularly noteworthy is their use in fluid flow simulations, where they address the challenges posed by traditional methods, especially in dealing with complex geometries and uncertain boundary conditions.[5–8] Fluid flow simulations play a crucial role in modeling various physical phenomena, including weather, climate, aerodynamics, and plasma physics.[9–11] Flow simulations play a crucial role in modeling various physical phenomena, including weather, climate, aerodynamics, and plasma physics.[9–11] The theoretical foundation of fluid mechanics relies on deriving governing equations from conservation laws, leading to partial differential equations like the well-established Navier–Stokes equations for Newtonian fluids.[6] Despite the availability of various computational fluid dynamics (CFD) methods, including finite difference (FDM), finite element (FEM), finite volume (FVM), and spectral element method (SEM),[12] the intricate flow fields of fluids often demand high-fidelity computational methods. However, these traditional numerical methods come with challenges such as complex computer codes, high computational costs required to resolve all physical scales, and uncertainties in parameter values.[13] In response to these challenges, physics-based data-driven techniques, such as PINNs, emerge as valuable tools. These innovative techniques play a crucial role in bridging the knowledge gap between observed physical phenomena and modeled physics.[13,14] A significant advancement was the introduction of "hidden fluid mechanics"[6] for solving ill-posed inverse fluid mechanics problems, using incompressible Navier–Stokes equations. Furthermore, efforts in directly simulating turbulence have led to the development of Navier–Stokes flow nets (NSFnets) that encompass both velocity–pressure and vorticity-velocity forms.[12] Some researchers have also simplified the Navier–Stokes equations, focusing on learning simplified Burgers' equations within a physical constraint framework.[15] To enhance accuracy, innovative neural network architectures have been explored. These include the adoption of spline convolutional neural networks[16] and the incorporation of variational auto-encoders.[17] Notably, some studies have effectively solved Reynolds-averaged Navier–Stokes equations by integrating large-eddy simulation data, constrained neural network training methods, and other advanced techniques.[15,18,19] These developments underscore the growing significance of PINNs in advancing computational fluid dynamics, showcasing their ability to tackle complex fluid flow simulations with increased efficiency and accuracy.

Despite the promising early results, PINNs do exhibit certain notable limitations, which warrant further investigation and improvement. One significant critique commonly raised against current deep learning algorithms, including PINNs, revolves around their interpretability and theoretical convergence properties, which remain inadequately understood. Moreover, an acknowledged drawback of PINNs pertains to their accuracy and computational efficiency. To enhance their accuracy, various strategies have been proposed. For instance, adaptive refinement based on equation residuals[20] and importance sampling techniques[21] have been suggested to improve the accuracy of PINNs. These strategies align with the observation that selecting training points near regions with steep gradients can enhance accuracy.[22] Researchers have suggested employing adaptive refinement techniques based on the residuals of equations[20] and importance sampling[21,23] to improve accuracy in specific regions characterized by steep gradients. Furthermore, unstable and unbalanced gradients across different loss components during model training have been identified as fundamental sources of failure in PINN methods.[24] To mitigate this issue, several approaches have been put forward to balance the diverse loss terms associated with the PDEs and initial/boundary conditions. These include learning rate annealing algorithms[24] and adaptive weight strategies leveraging the neural tangent kernel.[25]

To reduce the complexity and enhance the efficiency of PINN algorithms, domain decomposition has emerged as an increasingly popular approach. Domain decomposition methods follow a "divide and conquer" strategy by solving subproblems defined on smaller subdomains to tackle a problem on a global domain.[26] These methods offer the advantage of straightforward application to parallel computing, thereby improving computational efficiency. Various techniques have been proposed to incorporate domain decomposition into the PINN algorithm. For instance, Kharazmi *et al.*[27] introduced a general framework called hp-variational PINNs, which incorporates hp-refinement through domain decomposition. In this approach, the neural network is defined over the entire domain, while piecewise polynomials form the test space, enabling localized optimization of network parameters. Jagtap *et al.*[28–30] divided the computational domain into subdomains, using separate neural networks for each and incorporating interface terms in the loss function for consistency. To ensure consistency across subdomains, interface information is communicated and matched by incorporating additional interface terms in the loss function. Similarly, Li *et al.*[31] utilized neural networks to solve the subproblems of the Schwarz alternating method, while Dwivedi *et al.*[32] employed extreme learning machines in a distributed framework. Drawing inspiration from classical finite element methods, Moseley *et al.*[33] approximated the solution of differential equations by employing a sum of basis functions learned by neural networks defined over small and overlapping subdomains.

In addition to enhancing parallelization capacity, PINN algorithms based on domain decomposition methods also offer the advantage of improving accuracy. By employing separate neural networks to learn the subproblems within each subdomain,[28–34] this strategy can significantly enhance accuracy in several key aspects. First, the hyperparameters of individual neural networks in each subdomain can be properly adjusted based on prior knowledge. This allows a customized approach, capturing the characteristics of each subdomain effectively and leading to improved accuracy. Second, this strategy allows for the utilization of individualized and powerful neural networks in regions with high gradients. By doing so, errors are confined to the local region, preventing their propagation throughout the domain and improving the overall accuracy. This approach aligns with the principle of the properly-designed training point sampling near steep gradient regions.[20–22] Third, as the computational domain expands, the complexity of the solution tends to increase. This necessitates a more complex neural network and a larger number of training points, leading to a stiffer optimization problem. However, the neural networks used to approximate solutions within subdomains typically require less expressive power compared to those used for the global solution. This alleviates the stiffness of the optimization problem and contributes to enhanced accuracy, particularly for PDEs with complex solutions. Another factor contributing to the improved performance is the inclusion of additional continuity conditions at the interfaces, such as

residual continuity conditions, flux continuity, and higher-order continuity conditions.[28–30] In summary, PINN algorithms incorporating domain decomposition methods not only enhance parallelization capabilities but also improve accuracy.[35]

In this paper, we focus on the performance of this method in solving incompressible Navier–Stokes equations, examining factors like accuracy, convergence, and the impact of various strategies. We introduce innovative continuity conditions at interfaces, such as residual and flux continuity, and analyze their influence on predictive accuracy and interface consistency. Additionally, we propose two novel strategies: the dynamic weight method and a neural network architecture incorporating attention mechanisms. These are designed to address gradient pathologies commonly encountered in domain decomposition PINN methods. To validate our approach, we apply it to a range of problems, from standard benchmarks like the two-dimensional (2D) Kovasznay flow and the three-dimensional (3D) Beltrami flow to complex 3D blood flow simulations in synthetic and real blood vessel geometries. Our results demonstrate the method's efficacy in both forward and inverse problem scenarios, offering insights into its potential in fluid dynamics research. While this paper does not delve into the specifics of the distributed system implementation or parallel efficiency, it provides a comprehensive evaluation of the PINN method with domain decomposition, setting the stage for future research in this promising area.

In this paper, we distinguish our domain decomposition PINN approach by focusing exclusively on incompressible Navier–Stokes equations. While prior works focus on problems with large domains and multi-scale solutions,[33] elliptic problems,[31,34] and general equations.[27–30] In our treatment of subdomains, we employ individual networks within each subdomain, akin to Refs. 28–31 and 34, providing a straightforward application to parallel computing. However, we do not explore distributed system implementation specifics or parallel efficiency, differing from the works in Ref. 30. Moseley *et al.*[33] and Kharazmi *et al.*,[27] respectively, confine the solution to each subdomain by employing a window function and a test space containing piecewise polynomials. When addressing interface consistency, the mathematical construction of the PINN solution in Refs. 27 and 33 inherently ensures interface consistency. However, achieving parallelization might not be straightforward when employing a single PINN.[27] In contrast, interface information exchange is employed in Refs. 31 and 34. Meanwhile, works such as Refs. 28–30 enforce the average solution and flux continuity at the common interface. In addition to the average solution and flux continuity, our work goes further by considering higher-order continuity conditions, including residual continuity and the vanishing derivatives of the PDE residual. In summary, our approach to dealing with subdomains aligns with works such as Refs. 28–31 and 34, employing individual networks in each subdomain. Our distinct focus solves the Navier–Stokes equations by domain decomposition PINN approaches, examining factors like accuracy, convergence, and the impact of various strategies. More specifically, our contributions extend to a thorough investigation of various interface conditions, along with the introduction of two novel strategies. Our findings shed light on their significant impact on accuracy and interface consistency.

The rest of the paper is organized as follows. In Sec. II, the problem setup is introduced. Section III presents the methodology used in this paper, where we begin with a concise overview of PINNs and their application with domain decomposition for solving the Navier–Stokes equations. Additionally, we discuss approaches aimed at mitigating gradient pathologies. Section IV presents numerical experiments to validate the proposed method. Finally, Sec. V summarizes the findings and provides a discussion.

II. PROBLEM SETUP

The velocity–pressure (VP) form of the unsteady incompressible 3D Navier–Stokes equations is given by[1]

$$\frac{\partial \boldsymbol{u}}{\partial t} + \lambda_1(\boldsymbol{u}\cdot\nabla)\boldsymbol{u} = -\nabla p + \lambda_2\nabla^2\boldsymbol{u} \quad \text{in } \Omega\times[0,\,T], \tag{1a}$$

$$\nabla\cdot\boldsymbol{u} = 0 \quad \text{in } \Omega\times[0,\,T], \tag{1b}$$

$$\boldsymbol{u} = \boldsymbol{u}_\Gamma \quad \text{on } \Gamma_D\times[0,\,T], \tag{1c}$$

$$\frac{\partial \boldsymbol{u}}{\partial n} = 0 \quad \text{on } \Gamma_N\times[0,\,T], \tag{1d}$$

$$\boldsymbol{u}|_{t=0} = \boldsymbol{u}_0 \quad \text{in } \Omega, \tag{1e}$$

where the vector field $\boldsymbol{u}(\boldsymbol{x},t) = (u,v,w)$ is the velocity and u, v, and w are the three components of the velocity. The variables $\boldsymbol{x} = (x,y,z)$ and t denote the spatial and the temporal coordinates, respectively. T is the time period of the simulation. The scalar field p represents the pressure, while λ_1 and λ_2 are the physical properties of the fluid. Equation (1a) corresponds to the conservation of momentum, while Eq. (1b) represents the conservation of mass. Additionally, Eqs. (1c) and (1d) describe boundary conditions and (1e) is the initial condition where $\boldsymbol{u}_\Gamma$ and $\boldsymbol{u}_0$ are two given functions. Here, Ω, Γ_D, and Γ_N represent the computational domain, the Dirichlet boundary, and the Neumann boundary, respectively. Symbol definitions and units used in this paper are provided in Table I.

In this paper, the Navier–Stokes equations are solved using deep neural networks (NN) to get an approximation of the solution. The neural networks take the spatial and temporal coordinates as inputs and provide predictions for the pressure and velocity fields, i.e., mapping (x,y,z,t) to (u,v,w,p). The neural network serves as a surrogate model for the mathematical equations, and its parameters are optimized using various training data points. These include the initial and boundary conditions, the residual of the given equation (representing the errors of the momentum equations and the mass conservation equation), as well as observed data in the context of inverse problems.

III. METHODOLOGY

In this section, we present a concise overview of the physics-informed neural networks (PINNs), which were introduced in the work by Raissi *et al.*[1] We then delve into the PINNs with domain decomposition for solving the Navier–Stokes equations, highlighting key aspects such as subdomain loss functions, the various forms of interface conditions, and optimization methods. Furthermore, we introduce approaches aimed at mitigating gradient pathologies, including the use of the dynamic weight strategy and a neural network architecture incorporating attention mechanisms and residual connections.

A. Physics-informed neural networks

Let u^{NN} be a feed-forward neural network consisting of L hidden layers, depicted in Fig. 1(a). The network can be expressed as follows:

TABLE I. Symbol definitions and units. The subscript d represents the dth subdomain.

Symbol	Definition (units)
x	Spatial coordinate in the x-direction
y	Spatial coordinate in the y-direction
z	Spatial coordinate in the z-direction
x	Spatial coordinates
t	Time (s)
u	Velocity in the x-direction (m/s)
v	Velocity in the y-direction (m/s)
w	Velocity in the z-direction (m/s)
u	Velocity
p	Pressure (Pa)
∂	Partial derivative
∇	Vector differential operator
λ_1	Parameter of the Navier–Stokes equations
λ_2	Parameter of the Navier–Stokes equations
Ω	Computational domain (m^3)
Γ_D	Dirichlet boundary (m^3)
Γ_N	Neumann boundary (m^3)
Re	Reynolds number
ρ	Density of the blood flow (kg/m^3)
μ	Dynamic viscosity of the blood flow (Pa s)
ε_u	Relative L_2 error of the velocity in the x-direction
ε_v	Relative L_2 error of the velocity in the y-direction
ε_w	Relative L_2 error of the velocity in the z-direction
$\varepsilon_{\boldsymbol{u}}$	Relative L_2 error of the velocity
$\lvert\boldsymbol{u}\rvert$	Magnitude of the velocity (m/s)
$\varepsilon_{\lvert\boldsymbol{u}\rvert}$	Relative L_2 error of magnitude of the velocity
u^{NN}	Output of a neural network
Θ	Parameters of a neural network
σ	Activation function
$\mathscr{L}_d$	Loss function for the dth subdomain
$\mathscr{L}_d^{\mathrm{bc}}$	Boundary condition in the loss function
$\mathscr{L}_d^{\mathrm{ini}}$	Initial condition in the loss function
$\mathscr{L}_d^{\mathrm{re}}$	Residual of the governing equations in the loss function
$\mathscr{L}_d^{\mathrm{avg}}$	Discrepancy of predicted solutions across neighboring subdomains in the loss function
$\mathscr{L}_d^{\mathrm{conti}}$	Continuity condition along the interface in the loss function
N_d^{bc}	Number of selected boundary points
N_d^{int}	Number of selected interface points
N_d^{ini}	Number of selected initial points
N_d^{re}	Number of selected residual points
W_d^{bc}	Weight assigned to the boundary condition in the loss function
W_d^{ini}	Weight assigned to the initial condition in the loss function
W_d^{avg}	Weight assigned to the discrepancy of predicted solutions across neighboring subdomains in the loss function
W_d^{conti}	Weight assigned to the continuity condition along the interface in the loss function
W_d^{u}	Weight assigned to discrepancy between the predicted solutions and the observed data in the loss function
W_d^{λ}	Weight assigned to the discrepancy of predicted parameters across subdomains in the loss function

$$u^{NN}(\boldsymbol{z};\Theta) = T^{(L)} \circ T^{(L-1)} \cdots \circ T^{(0)}(\boldsymbol{z}), \tag{2}$$

where the symbol $\circ$ denotes the composition of functions, and $T^{(i)}$ represents the mapping in the ith layer. Here, each layer's mapping is represented by

$$\boldsymbol{z}_{i+1} := T^{(i)}(\boldsymbol{z}_i) = \sigma^{(i)}(W_i \boldsymbol{z}_i + b_i), \tag{3}$$

where $\boldsymbol{z}_i$ denotes the output vector at the $(i-1)$ th layer (i.e., the input vector at the ith layer), and $\boldsymbol{z}_0 = \boldsymbol{z}$ represents the input vector. The parameter set of the network is denoted as $\Theta = \{W_i, b_i\}_{i=0}^{L}$, with W_i representing the weight matrix and b_i denoting the bias vector of the ith layer. Generally, $\sigma^{(i)}$ is a non-linear mapping for $i = 0, \ldots, L-1$ and an identity mapping in the last layer. For this paper, we choose the hyperbolic tangent (tanh) function as the non-linear mapping $\sigma^{(i)}$. It is worth noting that the size of a feed-forward neural network with L hidden layers and W neurons in each hidden layer is denoted as $L \times W$.

In PINNs, a feed-forward neural network is employed to approximate the solutions of the equations. Specifically, for the Navier–Stokes equations, the neural network takes spatial and temporal coordinates (x, y, z, t) as inputs and predicts the solution (u, v, w, p). Here, we represent the neural network as $u^{NN}(\boldsymbol{x}, t; \Theta)$, with $\boldsymbol{x} = (x, y, z) \in \Omega$. The PINN model consists of a feed-forward component and a residual network responsible for calculating the residual of the governing equations. To obtain derivatives with respect to the space and time and consequently the residual of the equations, the chain rule is applied on u^{NN} using automatic differentiation, which is readily available in modern deep learning frameworks. To approximate the solution of the equations, the neural network $u^{NN}(\boldsymbol{x}, t; \Theta)$ is trained to satisfy the initial and boundary conditions, as well as the residual of the governing equations. This is achieved by formulating a minimization problem, where the loss function encompasses these conditions. For the Navier–Stokes equations given by (1a)–(1d), the loss function $\mathscr{L}(\Theta)$ can be defined as follows:

$$\mathscr{L}(\Theta) = \mathscr{L}^{\mathrm{re}}(\Theta) + W^{\mathrm{bc}}\mathscr{L}^{\mathrm{bc}}(\Theta) + W^{\mathrm{ini}}\mathscr{L}^{\mathrm{ini}}(\Theta), \tag{4}$$

where $\mathscr{L}^{\mathrm{ini}}$, $\mathscr{L}^{\mathrm{bc}}$, and $\mathscr{L}^{\mathrm{re}}$ represent the loss components corresponding to the initial conditions, the boundary conditions, and the residual of the governing equations, respectively. W^{bc} and W^{re} represent the weights assigned to the boundary and the residual terms, respectively. More specifically, the loss components can be defined as

$$\mathscr{L}^{\mathrm{ini}}(\Theta) = \frac{1}{N^{\mathrm{ini}}} \sum_{n=1}^{N^{\mathrm{ini}}} |\tilde{\boldsymbol{u}}(\boldsymbol{x}_n^{\mathrm{ini}}, 0) - \boldsymbol{u}(\boldsymbol{x}_n^{\mathrm{ini}}, 0)|^2, \tag{5}$$

$$\mathscr{L}^{\mathrm{bc}}(\Theta)=\frac{1}{N^{\mathrm{bc}}}\sum_{n=1}^{N^{\mathrm{bc}}}|\tilde{\boldsymbol{u}}(\boldsymbol{x}_n^{\mathrm{bc}},t_n^{\mathrm{bc}})-\boldsymbol{u}(\boldsymbol{x}_n^{\mathrm{bc}},t_n^{\mathrm{bc}})|^2, \tag{6}$$

$$\mathscr{L}^{\mathrm{re}}(\Theta)=\frac{1}{N^{\mathrm{re}}}\sum_{k=1}^{4}\sum_{n=1}^{N^{\mathrm{re}}}|f_k(\boldsymbol{x}_n^{\mathrm{re}},t_n^{\mathrm{re}})|^2, \tag{7}$$

where $\boldsymbol{u}$ and $\tilde{\boldsymbol{u}}$ represent the reference velocities and the velocities predicted by PINN, respectively. $\{(\boldsymbol{x}_n^{\mathrm{ini}},0)\}_{n=1}^{N^{\mathrm{ini}}}$, $\{(\boldsymbol{x}_n^{\mathrm{bc}},t_n^{\mathrm{bc}})\}_{n=1}^{N^{\mathrm{bc}}}$, and $\{(\boldsymbol{x}_n^{\mathrm{re}},t_n^{\mathrm{re}})\}_{n=1}^{N^{\mathrm{re}}}$ represent the sets of randomly selected initial, boundary, and residual points, respectively. Furthermore, according to (1a) and (1b), the residual of the governing equations, denoted as $\{f_k\}_{k=1}^4$ in (7), can be expressed as

$$\begin{cases} f_1=\dfrac{\partial u}{\partial t}+\lambda_1\left(u\dfrac{\partial u}{\partial x}+v\dfrac{\partial u}{\partial y}+w\dfrac{\partial u}{\partial z}\right)+\dfrac{\partial p}{\partial x} \\ \qquad -\lambda_2\left(\dfrac{\partial^2 u}{\partial x^2}+\dfrac{\partial^2 u}{\partial y^2}+\dfrac{\partial^2 u}{\partial z^2}\right), \\ f_2=\dfrac{\partial v}{\partial t}+\lambda_1\left(u\dfrac{\partial v}{\partial x}+v\dfrac{\partial v}{\partial y}+w\dfrac{\partial v}{\partial z}\right)+\dfrac{\partial p}{\partial y} \\ \qquad -\lambda_2\left(\dfrac{\partial^2 v}{\partial x^2}+\dfrac{\partial^2 v}{\partial y^2}+\dfrac{\partial^2 v}{\partial z^2}\right), \\ f_3=\dfrac{\partial w}{\partial t}+\lambda_1\left(u\dfrac{\partial w}{\partial x}+v\dfrac{\partial w}{\partial y}+w\dfrac{\partial w}{\partial z}\right)+\dfrac{\partial p}{\partial z} \\ \qquad -\lambda_2\left(\dfrac{\partial^2 w}{\partial x^2}+\dfrac{\partial^2 w}{\partial y^2}+\dfrac{\partial^2 w}{\partial z^2}\right), \\ f_4=\dfrac{\partial u}{\partial x}+\dfrac{\partial v}{\partial y}+\dfrac{\partial w}{\partial z}. \end{cases} \tag{8}$$

By defining the loss function $\mathscr{L}(\Theta)$ as shown in (4), the optimal parameters of the PINN model can be obtained by minimizing $\mathscr{L}(\Theta)$ using an optimization method.

B. Domain decomposition

In this paper, we investigate domain decomposition approaches within the framework of PINNs for solving the Navier–Stokes equations, which we refer to as NS-DDPINN (Navier–Stokes domain decomposed PINN). In NS-DDPINN, we divide the overall computational domain Ω into N_d non-overlapping subdomains, denoted as $\{\Omega_d\}_{d=1}^{N_d}$. For each subdomain, a separate neural network is employed to compute the local solution within that specific subdomain. These subdomains are interconnected through interface boundary conditions, facilitating communication and information exchange between adjacent subdomains. Consequently, in addition to incorporating the initial and boundary conditions, as well as the residual of the equations, into the PINN loss function, the loss function of each subdomain also includes terms related to the enforcement of interface conditions. The loss function for the dth subdomain, denoted as $\mathscr{L}_d(\Theta_d)$, can be expressed as follows:

$$\begin{aligned}\mathscr{L}_d(\Theta_d)=\mathscr{L}_d^{\mathrm{re}}(\Theta_d)+W_d^{\mathrm{bc}}\mathscr{L}_d^{\mathrm{bc}}(\Theta_d)+W_d^{\mathrm{ini}}\mathscr{L}_d^{\mathrm{ini}}(\Theta_d) \\ +W_d^{\mathrm{avg}}\mathscr{L}_d^{\mathrm{avg}}(\Theta_d)+\underbrace{W_d^{\mathrm{conti}}\mathscr{L}_d^{\mathrm{conti}}(\Theta_d)}_{\text{Interface condition}}.\end{aligned} \tag{9}$$

Here, the first three terms have the same definition as in (4), with the subscript d representing the dth subdomain. The last two terms correspond to the interface conditions used for communication between adjacent subdomains. The term $\mathscr{L}_d^{\mathrm{avg}}(\Theta_d)$ represents the mean squared error between the solutions of the current subdomain and its z subdomains. It is given by

$$\mathscr{L}_d^{\mathrm{avg}}(\Theta_d)=\sum_{\forall d^+}\frac{1}{N_d^{\mathrm{int}}}\sum_{n=1}^{N_d^{\mathrm{int}}}|\tilde{\boldsymbol{u}}_d(\boldsymbol{x}_{d,n}^{\mathrm{int}},t_{d,n}^{\mathrm{int}})-\tilde{\boldsymbol{u}}_{d^+}(\boldsymbol{x}_{d,n}^{\mathrm{int}},t_{d,n}^{\mathrm{int}})|^2, \tag{10}$$

where $\{(\boldsymbol{x}_{d,n}^{\mathrm{int}},t_{d,n}^{\mathrm{int}})\}_{n=1}^{N_d^{\mathrm{int}}}$ is the set of the randomly selected interface points, and the subscript d^+ represents the neighboring subdomain(s) of the dth subdomain.

In addition to the loss $\mathscr{L}_d^{\mathrm{avg}}$, we impose additional interface conditions, denoted by $\mathscr{L}_d^{\mathrm{conti}}$ in Eq. (9), to enhance the propagation of information throughout neighboring subdomains. These interface conditions consist of three types of solution continuity conditions: residual continuity, vanishing derivatives of the PDE residual, and flux continuity, that is, $\mathscr{L}_d^{\mathrm{conti}}=\mathscr{L}_d^{\mathrm{Ire}}+\mathscr{L}_d^{\mathrm{Igrad}}+\mathscr{L}_d^{\mathrm{Iflux}}$, where $\mathscr{L}_d^{\mathrm{Ire}}$, $\mathscr{L}_d^{\mathrm{Igrad}}$, are $\mathscr{L}_d^{\mathrm{Iflux}}$ will be defined later. The residual continuity condition aims to enforce consistency between adjacent subdomains by minimizing the difference in PDE residuals. The corresponding loss function $\mathscr{L}_d^{\mathrm{Ire}}(\Theta_d)$ is given by

$$\mathscr{L}_d^{\mathrm{Ire}}(\Theta_d)=\sum_{\forall d^+}\frac{1}{N_d^{\mathrm{int}}}\sum_{k=1}^{4}\sum_{n=1}^{N_d^{\mathrm{int}}}|f_{d,k}(\boldsymbol{x}_{d,n}^{\mathrm{int}},t_{d,n}^{\mathrm{int}})-f_{d^+,k}(\boldsymbol{x}_{d,n}^{\mathrm{int}},t_{d,n}^{\mathrm{int}})|^2, \tag{11}$$

where $\{f_{d,k}\}_{k=1}^4$ represents the residuals of the governing equations as defined by Eq. (8).

The continuity condition related to gradient information of the PDE residual is based on the observation that if the PDE residual is zero, its gradient should also be zero.[36] Therefore, we can enforce vanishing derivatives of the PDE residual at the common interface. The loss function $\mathscr{L}_d^{\mathrm{Igrad}}(\Theta_d)$ is given by

$$\mathscr{L}_d^{\mathrm{Igrad}}(\Theta_d)=\sum_{\forall d^+}\frac{1}{N_d^{\mathrm{int}}}\sum_{k=1}^{4}\sum_{n=1}^{N_d^{\mathrm{int}}}\sum_{x'\in\{x,y,z,t\}}|\partial_{x'}f_{d,k}(\boldsymbol{x}_{d,n}^{\mathrm{int}},t_{d,n}^{\mathrm{int}})|^2. \tag{12}$$

Finally, the flux continuity condition aims to ensure the consistency of fluxes across the common interface. The loss function $\mathscr{L}_d^{\mathrm{Iflux}}(\Theta_d)$ is given by

$$\mathscr{L}_d^{\mathrm{Iflux}}(\Theta_d)=\sum_{\forall d^+}\frac{1}{N_d^{\mathrm{int}}}\sum_{k=1}^{9}\sum_{n=1}^{N_d^{\mathrm{int}}}|\mathscr{F}_{d,k}(\boldsymbol{x}_{d,n}^{\mathrm{int}},t_{d,n}^{\mathrm{int}})-\mathscr{F}_{d^+,k}(\boldsymbol{x}_{d,n}^{\mathrm{int}},t_{d,n}^{\mathrm{int}})|^2, \tag{13}$$

where $\{\mathscr{F}_{d,k}\}_{k=1}^9$ represents the fluxes of the governing equations, as defined in Table II. It is important to note that the mass term of the fluxes is omitted in Eq. (13) since the C^0 continuity of (u, v, w) is enforced in the term $\mathscr{L}_d^{\mathrm{avg}}$. After optimizing the neural networks by minimizing the loss functions (9), each neural network provides the final solution in its corresponding subdomain. The solution values at the interfaces are set as the average of the solutions from the intersecting subdomains. Mathematically, the final solution is given by

$$u^{NN}(\boldsymbol{x},t;\{\Theta_d\}_{d=1}^{N_d})=\sum_{d=1}^{N_d}u_d^{NN}(\boldsymbol{x},t;\Theta_d)\delta_{\Omega_d}(\boldsymbol{x},t), \tag{14}$$

TABLE II. The fluxes of the governing equations.

Flux	x direction	y direction	z direction
x-momentum	$\lambda_1 u^2 + p - \lambda_2 \partial_x u$	$\lambda_1 uv - \lambda_2 \partial_y u$	$\lambda_1 uw - \lambda_2 \partial_z u$
y-momentum	$\lambda_1 uv - \lambda_2 \partial_x v$	$\lambda_1 v^2 + p - \lambda_2 \partial_y v$	$\lambda_1 vw - \lambda_2 \partial_z v$
z-momentum	$\lambda_1 uw - \lambda_2 \partial_x w$	$\lambda_1 vw - \lambda_2 \partial_y w$	$\lambda_1 w^2 + p - \lambda_2 \partial_z w$
Mass	u	v	w

where $u_d^{NN}(\boldsymbol{x}, t; \Theta_d)$ represents the output of the neural network corresponding to the dth subdomain. The function $\delta_{\Omega_d}(\boldsymbol{x}, t)$ is defined as follows:

$$\delta_{\Omega_d}(\boldsymbol{x}, t) = \begin{cases} 0 & \text{if } (\boldsymbol{x}, t) \notin \Omega_d, \\ 1 & \text{if } (\boldsymbol{x}, t) \in \Omega_d \backslash (\partial\Omega_d \backslash \Gamma_d), \\ \dfrac{1}{D} & \text{if } (\boldsymbol{x}, t) \in \partial\Omega_d \backslash \Gamma_d, \end{cases} \tag{15}$$

where $\partial\Omega_d \backslash \Gamma_d$, and Γ_d denote the common interface and the Dirichlet boundaries of the dth subdomain, respectively. D represents the number of subdomains intersecting along the common interface. Since no data are provided for the pressure during training, the variable p is considered as a hidden state and is obtained through the incompressibility constraint. To ensure consistency, a pressure shift is applied to the predicted pressure such that the mean pressure value matches between the reference solutions and the NS-DDPINN results. This pressure shift is implemented separately in each subdomain.

In the context of inverse problems, our goal is to simultaneously determine the unknown parameters λ_1 and λ_2 while achieving precise predictions of the velocity fields and pressure field, using scattered observed data. To achieve this, we utilize separate neural networks for each subdomain to learn the local solutions and parameters. The loss function can be written as

$$\begin{aligned} \mathscr{L}_d(\Theta_d) = \mathscr{L}_d^{\mathrm{re}}(\Theta_d) + W_d^u \mathscr{L}_d^u(\Theta_d) + W_d^{\mathrm{avg}} \mathscr{L}_d^{\mathrm{avg}}(\Theta_d) \\ + W_d^\lambda \mathscr{L}_d^\lambda(\Theta_d) + \underbrace{W_d^{\mathrm{conti}} \mathscr{L}_d^{\mathrm{conti}}(\Theta_d)}_{\text{Interface condition}}, \end{aligned} \tag{16}$$

where $\mathscr{L}_d^u$ measures the discrepancy between the neural network solutions and the observed data. It is computed as

$$\mathscr{L}_d^u(\Theta_d) = \frac{1}{N_d^u} \sum_{n=1}^{N_d^u} |\tilde{\boldsymbol{u}}_d(\boldsymbol{x}_{d,n}^u, t_{d,n}^u) - \boldsymbol{u}_d(\boldsymbol{x}_{d,n}^u, t_{d,n}^u)|^2, \tag{17}$$

where $\{(\boldsymbol{x}_{d,n}^u, t_{d,n}^u)\}_{n=1}^{N_d^u}$ represents the set of observed data points. In addition to the terms $\mathscr{L}_d^{\mathrm{avg}}$ and $\mathscr{L}_d^{\mathrm{conti}}$ that are similar to those used in forward problems, the loss function for inverse problems also includes a continuity condition for the parameters along the common interface. This condition ensures that the parameters λ_1 and λ_2 remain consistent across subdomains and is given by

$$\mathscr{L}_d^\lambda(\Theta_d) = \sum_{\forall d^+} \left(|\lambda_{1,d} - \lambda_{1,d^+}|^2 + |\lambda_{2,d} - \lambda_{2,d^+}|^2 \right), \tag{18}$$

where $\lambda_{1,d}$ and $\lambda_{2,d}$ represent the parameters λ_1 and λ_2 in the dth subdomain, respectively. It is worth noting that the unknown parameters λ_1 and λ_2 are treated as parameters of the PINNs and can be learned simultaneously with the network parameters using automatic differentiation.

In NS-DDPINN approaches, the training of neural networks can be parallelized when a sufficient number of processors are available. The training of each subdomain's network is independent of the others, except for the computation of the interface loss. To parallelize the NS-DDPINN method, a separate neural network is assigned to each subdomain and mapped to a dedicated processor, assuming there are enough processors available. During each iteration (or multiple iterations), the interface information is communicated between neighboring subdomains. Each processor computes its own loss function and updates the parameters of its corresponding model by minimizing the local loss function. This process is performed independently by each processor. As a result, the NS-DDPINN method can benefit from both model parallelism and data parallelism. It is important to note that in this study, we do not discuss the specific implementation of the NS-DDPINN method on distributed systems or the parallel efficiency of the method. These topics could be explored in future work to investigate the performance and scalability of the method in distributed computing environments.

C. Mitigating gradient pathologies

In PINN models, various forms of knowledge, including boundary conditions and PDE residuals, are incorporated by penalizing the loss functions that capture these aspects. However, training PINNs can be challenging due to unstable and unbalanced gradients between different components in the loss. This can result in failures for conventional PINN models. To overcome these issues, this section introduces two approaches to address these gradient pathologies: a dynamic weight method and a novel neural network architecture.

The composite loss functions in PINN models, such as (4) and (9), involve different terms that are weighted by regularization parameters like W^{bc} and W^{ini} in (4). However, during the training process of PINNs using gradient descent methods, the gradients of different components in the loss may vary significantly. In some cases, the term associated with boundary conditions may even suffer from vanishing gradient pathologies. It has been observed that the performance of PINN models heavily relies on the choices of these weighting coefficients, which can be tedious and problem-specific. To overcome this limitation, a dynamic weight method was proposed in Ref. 24. This method adaptively adjusts the weighting coefficients between different components in the loss by utilizing the back-propagated gradients during model training. Taking the loss function (4) as an example, the weighting coefficients at the $(k+1)$ th iteration are updated automatically as follows:

$$\begin{aligned} W^{\mathrm{bc}(k+1)} &= \alpha W^{\mathrm{bc}(k)} + (1-\alpha) \hat{W}^{\mathrm{bc}(k+1)}, \\ W^{\mathrm{ini}(k+1)} &= \alpha W^{\mathrm{ini}(k)} + (1-\alpha) \hat{W}^{\mathrm{ini}(k+1)}, \end{aligned} \tag{19}$$

where α is a hyperparameter (typically set to 0.9) that determines the contributions of the previous dynamic weights. The quantities $\mathscr{W}^{\mathrm{bc}(k+1)}$ and $\mathscr{W}^{\mathrm{ini}(k+1)}$ represent the relation between gradient statistics of different components. Specifically, they can be expressed as

$$\mathscr{W}^{\mathrm{bc}(k+1)} = \frac{\max_\Theta\{|\nabla_\Theta \mathscr{L}^{\mathrm{re}}|\}}{\overline{|\nabla_\Theta W^{\mathrm{bc}(k)} \mathscr{L}^{\mathrm{bc}}|}}, \quad \mathscr{W}^{\mathrm{ini}(k+1)} = \frac{\max_\Theta\{|\nabla_\Theta \mathscr{L}^{\mathrm{re}}|\}}{\overline{|\nabla_\Theta W^{\mathrm{ini}(k)} \mathscr{L}^{\mathrm{ini}}|}}, \tag{20}$$

where $\overline{|\nabla_\Theta W^{\mathrm{bc}(k)} \mathscr{L}^{\mathrm{bc}}|}$ and $\overline{|\nabla_\Theta W^{\mathrm{ini}(k)} \mathscr{L}^{\mathrm{ini}}|}$ denote the mean values of $|\nabla_\Theta W^{\mathrm{bc}(k)} \mathscr{L}^{\mathrm{bc}}|$ and $|\nabla_\Theta W^{\mathrm{ini}(k)} \mathscr{L}^{\mathrm{ini}}|$, respectively. Similarly, for PINNs with domain decomposition [e.g., using the loss function (9)], the weighting coefficients related to different components can be updated in the same way as (19). The dynamic weights method provides an automated approach to balance the interaction between different components in the loss, providing improved stability and performance during the training of PINN models.[24]

In addition to appropriately penalizing the loss function, another approach to address gradient pathologies in PINN models is through careful design of the neural network architecture. Recently, attention mechanisms and residual connections have shown significant performance improvements in various natural language processing tasks. Attention mechanisms allow models to dynamically emphasize important features through multiplicative interactions between different input dimensions. On the other hand, residual connections in neural networks enable information to bypass one or more layers, facilitating smoother information propagation and improved optimization. Empirical evidence has demonstrated that residual networks converge faster and achieve higher accuracy. Motivated by the success of attention mechanisms and residual connections, Ref. 24 proposed a novel neural network architecture that combines these two strategies. Building upon a conventional fully-connected architecture as the backbone, the proposed architecture introduces two additional branches to project the inputs into a high-dimensional feature space. As illustrated in Fig. 1(b), apart from the output of the backbone in the $(i-1)$ th layer $(i = 1, \ldots, L-1)$, the input variables are also connected to the ith layer. These inputs are fused into the input of the ith layer using a point-wise multiplication operation. Mathematically, the input of the ith layer is given by

$$\boldsymbol{h}_i := \mathscr{H}(\boldsymbol{z}_i, \boldsymbol{f}, \boldsymbol{h}) = \boldsymbol{z}_i \odot \boldsymbol{f} + (\boldsymbol{1} - \boldsymbol{z}_i) \odot \boldsymbol{g}, \quad i = 1, \ldots, L-1, \tag{21}$$

where $\odot$ denotes the point-wise multiplication operation and $\mathbf{1}$ is a vector or matrix with all entries equal to one. $\boldsymbol{f}$ and $\boldsymbol{g}$ are the inputs of the introduced branches, obtained by transforming the inputs $\mathbf{z}_0$ through two separate fully-connected layers. Specifically

$$\boldsymbol{f} = \sigma(\tilde{W}^1 \boldsymbol{z}_0 + \tilde{b}^1), \quad \boldsymbol{h} = \sigma(\tilde{W}^2 \boldsymbol{z}_0 + \tilde{b}^2), \tag{22}$$

where σ is the activation function, typically the hyperbolic tangent (tanh). Consequently, the output of the backbone in the ith layer can be written as

$$\boldsymbol{z}_{i+1} := T^{(i)}(\boldsymbol{h}_i) = \sigma^{(i)}(W_i \boldsymbol{h}_i + b_i), \quad i = 1, \ldots, L-1. \tag{23}$$

It is worth noting that the number of parameters in this new model are almost identical to those of a conventional fully-connected architecture, except for the additional weights and biases $\tilde{W}^1, \tilde{b}^1, \tilde{W}^2, \tilde{b}^2$ introduced for the added branches.

To evaluate the impact of these strategies, the dynamic weight method and the novel neural network architecture are applied in simulations to explore their effects on the convergence of the training process and the predictive accuracy.

IV. NUMERICAL EXPERIMENTS

In this section, we conduct several experiments to assess the performance of the PINN model with domain decomposition in simulating incompressible flows. Our investigations cover both forward and inverse problems in various scenarios, including the 2D Kovasznay flow, the 3D Beltrami flow, the 2D lid-driven cavity flow, the 2D cylinder wake, and the 3D blood flow. To perform these experiments, we utilize the TensorFlow framework and carry out the computations on a workstation equipped with a 1 NVIDIA Tesla V100 32G GPU. The neural networks are optimized using the Adam optimization algorithm, with their parameters randomly initialized using the Xavier initialization method. Subsequently, we employ the limited-memory Broyden–Fletcher–Goldfarb–Shanno Bound (L-BFGS-B) algorithm to fine-tune the neural networks and achieve higher accuracy. This fine-tuning process is automatically terminated based on a specified increment tolerance criterion. To evaluate the performance of the NS-DDPINN methods, we employ the relative L_2 error metric, which measures the discrepancy between the solutions predicted by the neural networks ($\tilde{\boldsymbol{u}}$ and $\tilde{p}$) and the reference solutions ($\boldsymbol{u}$ and p). Specifically, the relative L_2 error is calculated as follows:

$$\varepsilon_{\boldsymbol{u}} = \frac{\|\tilde{\boldsymbol{u}} - \boldsymbol{u}\|_2}{\|\boldsymbol{u}\|_2}, \quad \varepsilon_p = \frac{\|\tilde{p} - p\|_2}{\|p\|_2}, \tag{24}$$

where $\boldsymbol{u}$ and p represent the velocities (u, v, w) and pressure, respectively, while the tilde denotes the values inferred by the neural networks. Additionally, the magnitude of the velocity solution, denoted as $|\boldsymbol{u}(\boldsymbol{x})| = \sqrt{u^2(\boldsymbol{x}) + v^2(\boldsymbol{x}) + w^2(\boldsymbol{x})}$, is also considered for velocity comparisons.

A. Kovasznay flow

In the Kovasznay flow, the analytical solution for the 2D steady Navier–Stokes equations is provided as follows:

$$\begin{aligned} u(x,y) &= 1 - \exp(\xi x)\cos(2\pi y), \\ v(x,y) &= \frac{\xi}{2\pi}\exp(\xi x)\sin(2\pi y), \\ p(x,y) &= \frac{1}{2}(1 - \exp(2\xi x)), \end{aligned} \tag{25}$$

where the parameters are defined as

$$\xi = \frac{1}{2\nu} - \sqrt{\frac{1}{4\nu^2} + 4\pi^2}, \quad \nu = \lambda_2 = \frac{1}{40}, \quad \lambda_1 = 1.$$

Regarding the computational domain, we set it as $\Omega = [-0.5, 1.0] \times [-0.5, 1.5]$, which is decomposed into four subdomains by evenly dividing the x- and y-coordinates into two parts. Unless specified otherwise, the experimental settings are as follows: (1) each subdomain uses 51 points on each boundary (for both Dirichlet boundary and interface boundary), resulting in $N_d^{\mathrm{bc}} = 2 \times 51$ and $N_d^{\mathrm{int}} = 2 \times 51$ for $d \in \{1, 2, 3, 4\}$; (2) for residual points, 1000 points are randomly sampled inside each subdomain, resulting in

TABLE III. Kovasznay flow: Relative L_2 errors of the velocity and pressure solutions for NS-DDPINNs with varying depth and width of the neural networks. These errors represent the mean ± standard deviation of five different runs with independent initial network parameters.

Depth \ Width		25	50	100	200
2	$\varepsilon_u(\%)$	0.0956 ± 0.0326	0.0654 ± 0.0154	0.0820 ± 0.0185	0.0579 ± 0.0126
	$\varepsilon_v(\%)$	0.7053 ± 0.3218	0.4712 ± 0.1415	0.5765 ± 0.1065	0.4391 ± 0.1391
	$\varepsilon_p(\%)$	0.2716 ± 0.1027	0.1796 ± 0.0303	0.2127 ± 0.0508	0.1587 ± 0.0465
4	$\varepsilon_u(\%)$	0.0510 ± 0.0153	0.0454 ± 0.0104	0.0538 ± 0.0124	0.0413 ± 0.0132
	$\varepsilon_v(\%)$	0.3512 ± 0.0444	0.3543 ± 0.1158	0.3635 ± 0.1119	0.3031 ± 0.1082
	$\varepsilon_p(\%)$	0.1592 ± 0.0366	0.1225 ± 0.0433	0.1377 ± 0.0306	0.1060 ± 0.0254
8	$\varepsilon_u(\%)$	0.0437 ± 0.0177	0.0656 ± 0.0183	0.1601 ± 0.0376	0.5488 ± 0.0801
	$\varepsilon_v(\%)$	0.3412 ± 0.1411	0.4118 ± 0.1403	0.9794 ± 0.3550	2.5435 ± 0.3902
	$\varepsilon_p(\%)$	0.1169 ± 0.0326	0.1547 ± 0.0266	0.3665 ± 0.0980	1.1807 ± 0.1789

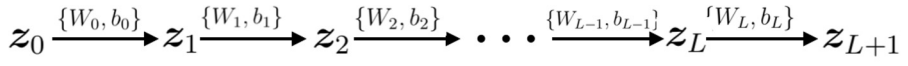
$N_d^{\text{re}} = 1000$; (3) a neural network architecture with four hidden layers and 50 neurons in each hidden layer (i.e., of size 4×50) is used for all subdomains; (4) the networks are optimized through 30 000 Adam iterations with a learning rate of 10^{-3}. Subsequently, L-BFGS-B training is employed for further optimization; (5) during the inference stage, a grid of 100×100 regularly sampled test points spanning the entire domain Ω is used to compute the error compared to the reference solutions.

1. Effect of the depth and width of the networks

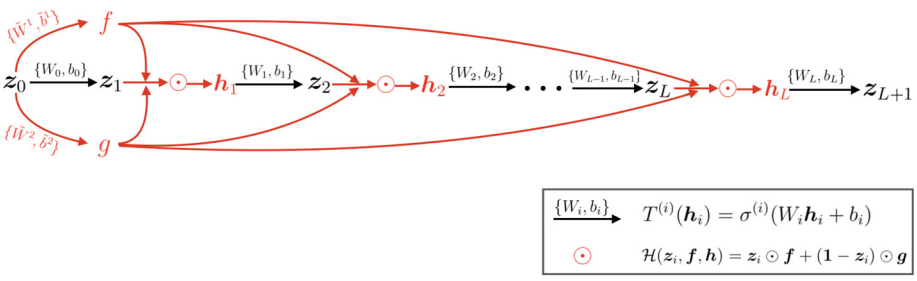
We begin by examining the impact of the neural network depth and width on the performance. For the interface condition in the loss functions, only the average solution term, $\mathscr{L}_d^{\text{avg}}$, is utilized. Additionally, fixed weighting coefficients of $W_d^{\text{bc}} = W_d^{\text{avg}} = 1$ are applied for both boundary and interface conditions. Table III presents the relative L_2 error of the solutions across the entire domain. The depth and width of the neural network are varied in the sets $\{2, 4, 8\}$ and $\{25, 50, 100, 200\}$, respectively. Increasing the size of the neural network enhances its capacity to capture complex patterns and improve the overall performance. However, this improvement comes at the cost of potential overfitting when there is insufficient training data. Table III provides valuable insights into this trade-off. The table reveals several key observations. First, within a certain range, increasing the size of the neural network improves the accuracy of solutions. However, it is important to note that further enlargement of the network can lead to overfitting. On the other hand, it is important to note that deeper networks are more prone to overfitting when the number of hidden neurons remains constant. This aligns with the growing consensus that deep networks offer greater expressivity compared to shallow networks of comparable size.

2. Dynamic weight strategy

The performance of the dynamic weight strategy is evaluated by comparing it with the fixed weight strategy. Figure 2 illustrates the evolution of dynamic weights for four subdomains, with initial weights set as $W_d^{\text{bc}} = W_d^{\text{avg}} = 1$ or $W_d^{\text{bc}} = W_d^{\text{avg}} = 100$. Notably, regardless of the initial values, the dynamic weights converge to similar values. The weights associated with boundary conditions (i.e., W_d^{bc}) converge to about 10–15, while the weights related to interface conditions (i.e., W_d^{avg}) converge to about 4–8. Figure 3 demonstrates the convergence



(a) Conventional fully-connected neural networks.



(b) Neural network with attention mechanisms.

FIG. 1. Architecture comparison between a conventional fully-connected neural network (a) and a neural network with attention mechanisms (b). The neural network with attention mechanisms builds upon the conventional fully-connected architecture, incorporating an additional attention mechanism highlighted in red.

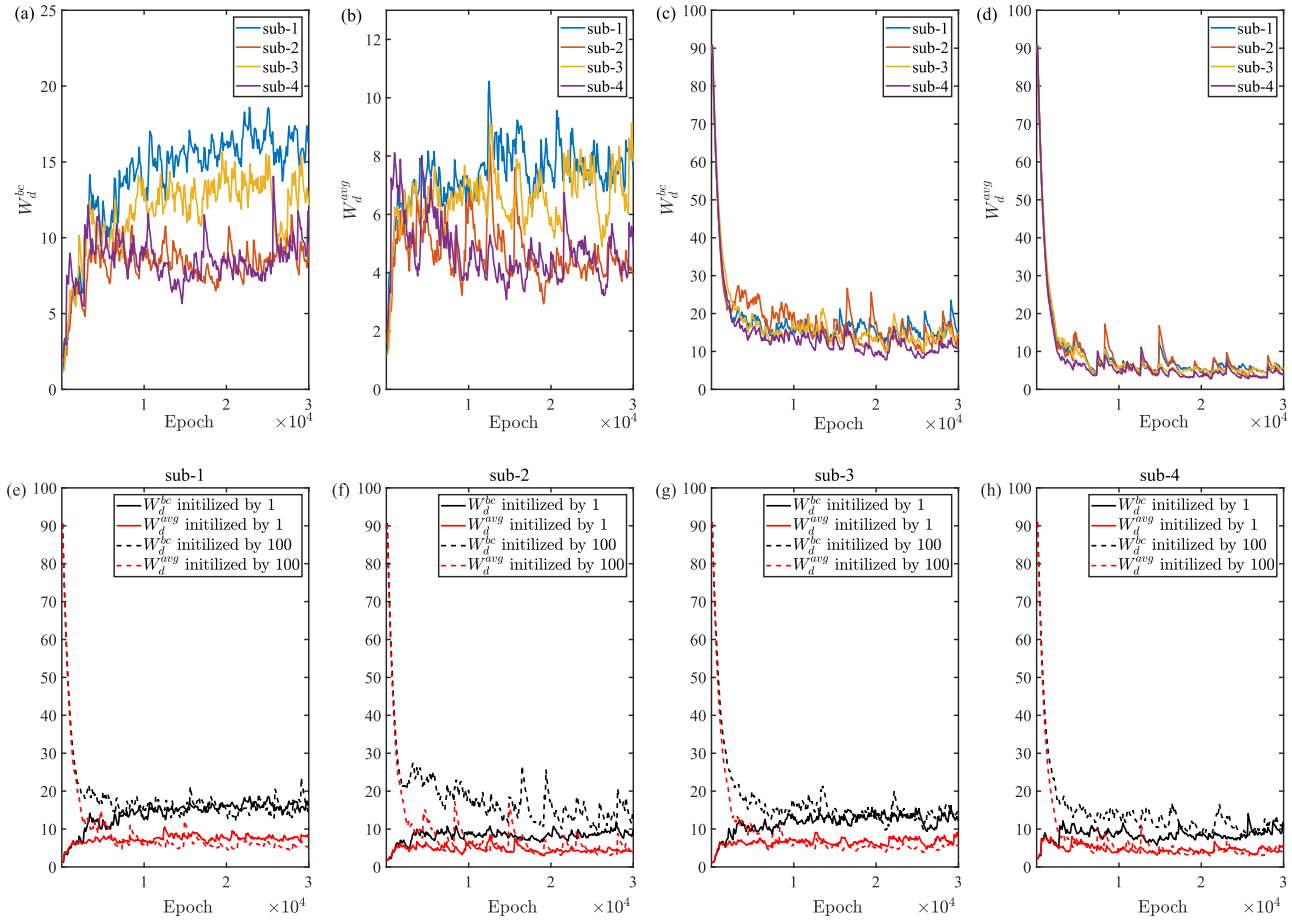


FIG. 2. Kovasznay flow: Evolution of dynamic weights with different initial weights for four subdomains. (a) and (b): initial weights $W_d^{\text{bc}} = W_d^{\text{avg}} = 1$; (c) and (d): initial weights $W_d^{\text{bc}} = W_d^{\text{avg}} = 100$; (e)–(h): different initial weights for four subdomains.

of the relative L_2 error with the number of residual points for different weight strategies. The results indicate that the dynamic weight strategy improves prediction accuracy, particularly when the initial weights are set to $W_d^{\text{bc}} = W_d^{\text{avg}} = 100$. Additionally, Fig. 4 presents histograms of the gradients of the loss function with respect to the parameters of the PINN models. It is observed that when fixed weights are used, the gradients of different loss components become unbalanced. For example, with fixed weights $W_d^{\text{bc}} = W_d^{\text{avg}} = 1$, the gradients of the loss function corresponding to the residuals dominate the overall gradients. Conversely, when fixed weights $W_d^{\text{bc}} = W_d^{\text{avg}} = 100$ are employed, the gradients of the loss function associated with the boundary and interface conditions dominate the overall gradients. Applying dynamic weights ensures a more consistent behavior of the gradients for different loss terms, improving the overall balance and stability of the optimization process.

3. Optional interface condition

We assess the impact of additional interface conditions in the loss functions on the performance of the models. Specifically, we consider the use of the average solution term of the interface condition, denoted as $\mathscr{L}_d^{\text{avg}}$, in all cases. We compare the performances of three optional additional interface conditions: residual continuity ($\mathscr{L}_d^{\text{Ire}}$), vanishing derivatives of the PDE residual ($\mathscr{L}_d^{\text{Igrad}}$), and flux continuity ($\mathscr{L}_d^{\text{Iflux}}$). Table IV presents the relative L_2 errors of velocity and pressure solutions for each case. Additionally, Fig. 5 provides a snapshot of the velocity and pressure fields, along with the absolute errors for all the cases. The results indicate that the inclusion of additional interface conditions enhances the accuracy of the predicted solutions. Residual continuity and vanishing derivatives of the PDE residual yield comparable accuracy. Notably, flux continuity achieves the highest accuracy and significantly improves continuity along the common interface. Furthermore, the utilization of the dynamic weight strategy further enhances the accuracy of the predicted solutions, achieving relative L_2 errors on the order of 10^{-5}.

B. 3D Beltrami flow

The unsteady 3D Beltrami flow has the following analytic solution:[37]

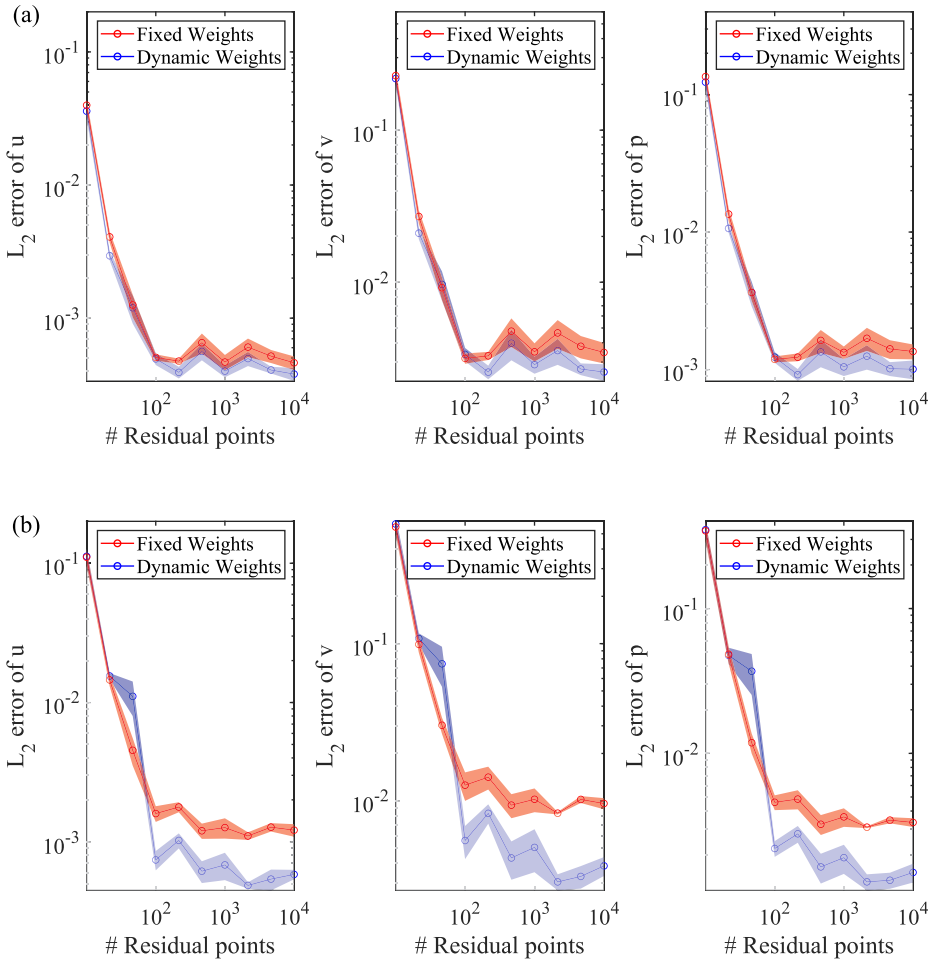


FIG. 3. Kovasznay flow: Convergence of the relative L_2 error with the number of residual points for different weight strategies: (a) with initial weights $W_d^{bc} = W_d^{avg} = 1$; (b) with initial weights $W_d^{bc} = W_d^{avg} = 100$. The term "Fixed wights" refers to using fixed weights during training, while "Dynamic weights" refer to using the dynamic weights given by (19). The shaded regions represent one standard deviation from five independent runs with independent initial network parameters.

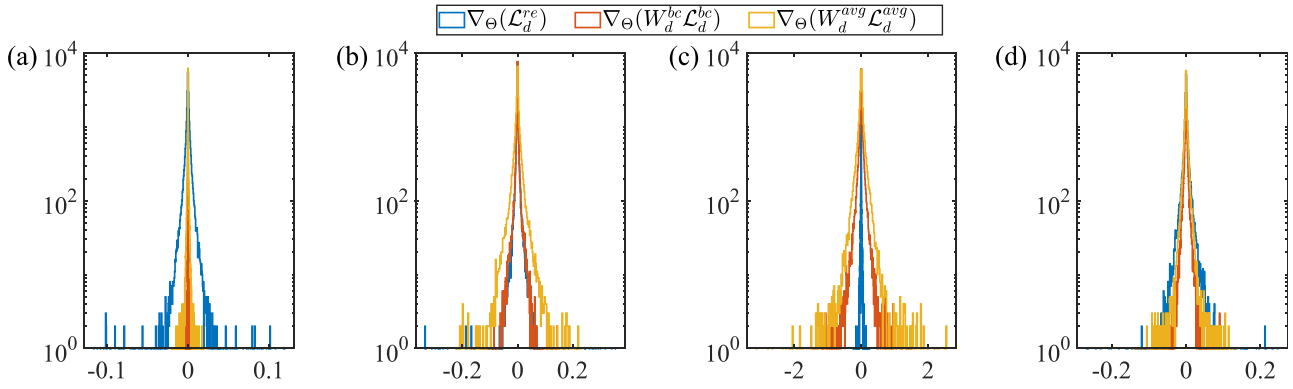


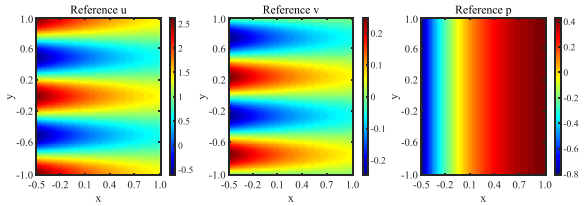
FIG. 4. Kovasznay flow: Histograms of the gradients of loss function associated with boundary conditions [i.e., $\nabla_\Theta(\mathcal{L}_d^{re})$], the residual of PDE [i.e., $\nabla_\Theta(W_d^{bc}\mathcal{L}_d^{bc})$], and interface conditions [i.e., $\nabla_\Theta(W_d^{avg}\mathcal{L}_d^{avg})$] after 20 000 iterations during training. Different weight strategies are compared: (a) fixed weight $W_d^{bc} = W_d^{avg} = 1$; (b) dynamic weight with initial weights $W_d^{bc} = W_d^{avg} = 1$; (c) fixed weight $W_d^{bc} = W_d^{avg} = 100$; (d) dynamic weight with initial weights $W_d^{bc} = W_d^{avg} = 100$.

TABLE IV. Kovasznay flow: Relative L_2 errors of velocity and pressure solutions for NS-DDPINNs with different additional interface conditions. The settings of additional interface conditions, from top to bottom, are: without additional interface conditions, residual continuity ($\mathscr{L}_d^{\text{Ire}}$), vanishing derivatives of the PDE residual ($\mathscr{L}_d^{\text{Igrad}}$), flux continuity ($\mathscr{L}_d^{\text{Iflux}}$), and the combination of flux continuity and dynamic weight strategy. These errors represent the mean ± standard deviation of five different runs with independent initial network parameters.

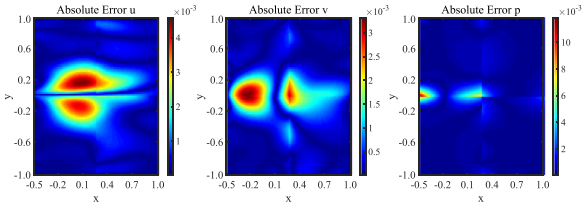
	$\varepsilon_u(\%)$	$\varepsilon_v(\%)$	$\varepsilon_p(\%)$
Without additional interface condition	0.1030 ± 0.0423	0.8338 ± 0.3788	0.2824 ± 0.1193
Residual continuity	0.0489 ± 0.0138	0.3846 ± 0.1081	0.1563 ± 0.0402
Vanishing derivatives of the residual	0.0453 ± 0.0080	0.3717 ± 0.0374	0.1375 ± 0.0054
Flux continuity	0.0150 ± 0.0025	0.1101 ± 0.0101	0.0526 ± 0.0124
Flux continuity + dynamic weights	0.0082 ± 0.0013	0.0708 ± 0.0082	0.0295 ± 0.0050

$$
\begin{aligned}
u &= -a\left[\exp(ax)\sin(ay+dz)+\exp(az)\cos(ax+dy)\exp(-d^2t)\right],\\
v &= -a\left[\exp(ay)\sin(az+dx)+\exp(ax)\cos(ay+dz)\exp(-d^2t)\right],\\
w &= -a\left[\exp(az)\sin(ax+dy)+\exp(ay)\cos(az+dx)\exp(-d^2t)\right],\\
p &= -\frac{1}{2}a^2[\exp(2ax)+\exp(2ay)+\exp(2az)\\
&\quad +2\sin(ax+dy)\cos(az+dx)\exp(a(y+z))]\\
&\quad +2\sin(ay+dz)\cos(ax+dy)\exp(a(z+x))\\
&\quad +2\sin(az+dx)\cos(ay+dz)\exp(a(x+y))]\exp(-2d^2t),
\end{aligned}
$$

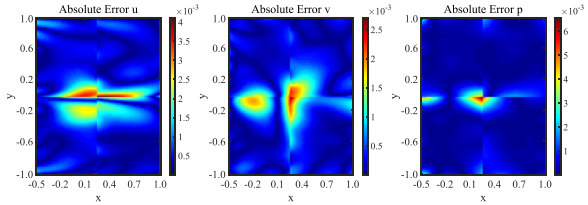
where $a = d = 1$ and $\lambda_1 = \lambda_2 = 1$. The computational domain Ω spans the spatial coordinates $[-1,1]\times[-1,1]\times[-1,1]$, and the time interval is $[0,1]$. To perform the experiments, the computational domain Ω is divided into eight subdomains, with the x-, y-, and z-coordinates evenly divided into two parts. The following experimental settings are used unless specified otherwise: (1) each subdomain's boundaries are discretized with 21×21 points on each face, and there are 11 regularly spaced time points, resulting in $N_d^{\text{bc}} = 3\times21\times21\times11$ and $N_d^{\text{int}} = 3\times21\times21\times11$ for $d\in\{1,\ldots,8\}$. Each subdomain has $21\times21\times21$ initial points, giving $N_d^{\text{ini}} = 21\times21\times21$.



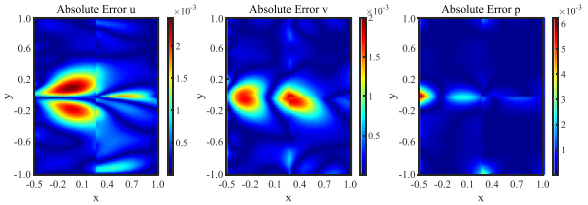
(a) Reference solutions



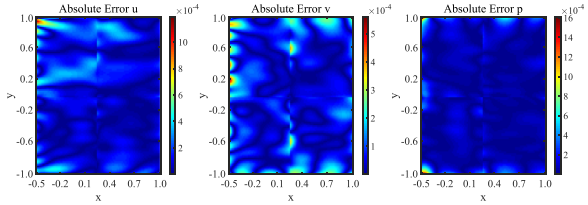
(b) Without additional interface condition, $(\varepsilon_u,\varepsilon_v,\varepsilon_p)(\%) = (0.1025, 0.7867, 0.2799)$



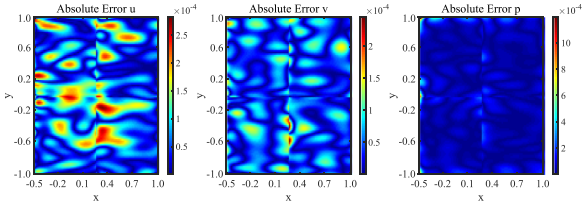
(c) Residual continuity, $(\varepsilon_u,\varepsilon_v,\varepsilon_p)(\%) = (0.0728, 0.5625, 0.2252)$



(d) Vanishing derivatives of PDE residual, $(\varepsilon_u,\varepsilon_v,\varepsilon_p)(\%) = (0.0532, 0.4398, 0.1568)$



(e) Flux continuity, $(\varepsilon_u,\varepsilon_v,\varepsilon_p)(\%) = (0.0133, 0.0979, 0.0368)$



(f) Flux continuity + dynamic weights, $(\varepsilon_u,\varepsilon_v,\varepsilon_p)(\%) = (0.0078, 0.0583, 0.0240)$

FIG. 5. Kovaszny flow: Velocity and pressure fields, along with the absolute errors, for different additional interface conditions. (a) Reference solutions; (b) without additional interface condition; (c) residual continuity; (d) vanishing derivatives of the PDE residual; (e) flux continuity; and (f) combining flux continuity and dynamic weight strategy.

TABLE V. Beltrami flow: Relative L_2 errors of velocity and pressure solutions for NS-DDPINNs at different time points and for the entire domain. The reported errors represent the mean ± standard deviation of five independent runs with distinct initial network parameters.

	$t=0$	$t=0.25$	$t=0.50$	$t=0.75$	$t=1.00$	Entire domain
$\varepsilon_u(\%)$	0.0281 ± 0.0010	0.0515 ± 0.0030	0.0571 ± 0.0054	0.0871 ± 0.0192	0.1616 ± 0.0365	0.0613 ± 0.0079
$\varepsilon_v(\%)$	0.0324 ± 0.0014	0.0538 ± 0.0048	0.0630 ± 0.0079	0.0971 ± 0.0256	0.1898 ± 0.0483	0.0687 ± 0.0101
$\varepsilon_w(\%)$	0.0337 ± 0.0021	0.0722 ± 0.0051	0.0908 ± 0.0090	0.1241 ± 0.0208	0.2128 ± 0.0390	0.0862 ± 0.0083
$\varepsilon_p(\%)$	0.3172 ± 0.0170	0.3078 ± 0.0188	0.4950 ± 0.0628	0.9602 ± 0.1169	2.1149 ± 0.0779	2.4360 ± 1.2204

For the residual points, 2000 points are randomly sampled within each spatiotemporal subdomain, resulting in $N_d^{\mathrm{re}} = 2000$; (2) a neural network architecture of size 7×50 is used for all subdomains; (3) the additional interface conditions include flux continuity. The weighting coefficients for the boundary, initial, and interface conditions are set as $W_d^{\mathrm{bc}} = W_d^{\mathrm{ini}} = W_d^{\mathrm{avg}} = 100$ and $W_d^{\mathrm{conti}} = 1$; (4) the networks are optimized through 30 000 Adam iterations with a learning rate of 10^{-3}, followed by L-BFGS-B training; (5) during the inference stage, a total of $51 \times 51 \times 51 \times 11$ regularly sampled test points are used throughout the spatiotemporal domain to compute the errors compared to the reference solutions.

Table V presents the relative L_2 errors of velocity and pressure solutions at various time points throughout the domain. The results demonstrate that the relative L_2 errors generally reach the order of 10^{-4} at most time points. In Fig. 6, a snapshot of the velocity and pressure fields, along with the corresponding absolute errors, is displayed at $t=1$ on the face $z=0$. The visualization reveals that the point-wise absolute errors can reach the order of 10^{-3}. Next, we compare the performance of PINNs with and without domain decomposition. To ensure a fair comparison, both cases maintain an equal number of residual points, boundary points, and initial points, as well as the same network size. Table VI presents the relative L_2 errors of velocity and pressure solutions for two different network sizes (4×50 and 7×50) in both cases. The results indicate that increasing the network size leads to higher accuracy in both cases. Furthermore, the PINN with domain decomposition consistently outperforms the one without domain decomposition. The improved performance of PINNs with domain decomposition can be attributed to several factors. First, the domain decomposition approach incorporates additional information regarding interface conditions, such as flux continuity. Moreover, by employing separate networks in each subdomain, the expressive power of the overall network is enhanced.

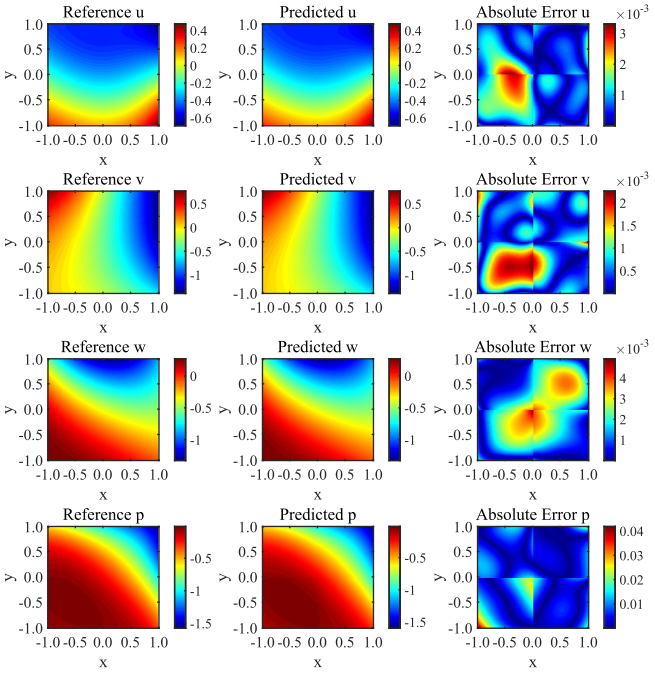


FIG. 6. Beltrami flow: The referenced and predicted solutions (velocity and pressure fields), along with the corresponding absolute errors, for NS-DDPINNs at $t=1$ and the face $z=0$.

C. Lid-driven cavity flow

The lid-driven cavity flow in a 2D square cavity is considered as a benchmark problem in the subsection. The computational domain is $\Omega = [0,1] \times [0,1]$, where the top boundary is driven with a constant velocity in the positive x direction. The other three sides enforce a no-slip boundary condition. Specifically, the boundary conditions are set to $(u,v) = (1,0)$ when $y=1$ and $(u,v) = (0,0)$ otherwise. Two cases are investigated, with different values of the parameter λ_2. In the first case, $\lambda_2 = 1/100$ is chosen, corresponding to a Reynolds number $\mathrm{Re} = 100$. In the second case, $\lambda_2 = 1/400$ is used, resulting in a Reynolds number $\mathrm{Re} = 400$.

The experimental settings are as follows: (1) The computational domain Ω is divided into four subdomains by evenly dividing the x- and y-coordinates into two parts; (2) For each subdomain, the Dirichlet boundaries are discretized with 80 points, resulting in a total of $N_d^{\mathrm{bc}} = 2 \times 80$ points. The interface boundaries between the subdomains are discretized with 250 points each, leading to $N_d^{\mathrm{int}} = 2 \times 250$ points. Residual points are randomly sampled within each subdomain, with a total of $N_d^{\mathrm{re}} = 12\,000$ points; (3) A neural network architecture of size 4×50 is employed in all subdomains; (4) The flux continuity is enforced in the additional interface conditions. Fixed weighting coefficients $W_d^{\mathrm{bc}} = W_d^{\mathrm{avg}} = W_d^{\mathrm{conti}} = 1$ are used for boundary and interface conditions; (5) The neural networks undergo optimization through 30 000 iterations using the Adam optimization algorithm. The learning rate is set initially as 10^{-3} and follows an exponentially decreasing schedule with a decay factor of 0.9 every 1000 epochs. Subsequently, the L-BFGS-B training is employed for further optimization; (6) In the inference stage, a regular grid of 101×101 test points is used throughout the domain to compute the error compared to the reference solutions from Ghia *et al.*[38]

TABLE VI. Beltrami flow: Relative L_2 errors of velocity and pressure solutions are compared between NS-DDPINN (PINN with the domain decomposition) and NS-PINN (PINN without the domain decomposition). The neural networks used have sizes of 4×50 and 7×50. The reported errors represent the mean $\pm$ standard deviation of five independent runs with distinct initial network parameters.

	NN	$\varepsilon_u(\%)$	$\varepsilon_v(\%)$	$\varepsilon_w(\%)$	$\varepsilon_p(\%)$
NS-DDPINN	4×50	0.0900 ± 0.0252	0.0861 ± 0.0117	0.0954 ± 0.0131	8.0686 ± 1.8272
	7×50	0.0613 ± 0.0079	0.0687 ± 0.0101	0.0862 ± 0.0083	2.4360 ± 1.2204
NS-PINN	4×50	0.1138 ± 0.0256	0.1081 ± 0.0111	0.1115 ± 0.0228	5.4147 ± 5.4783
	7×50	0.0885 ± 0.0140	0.0937 ± 0.0181	0.0972 ± 0.0176	6.9328 ± 5.9152

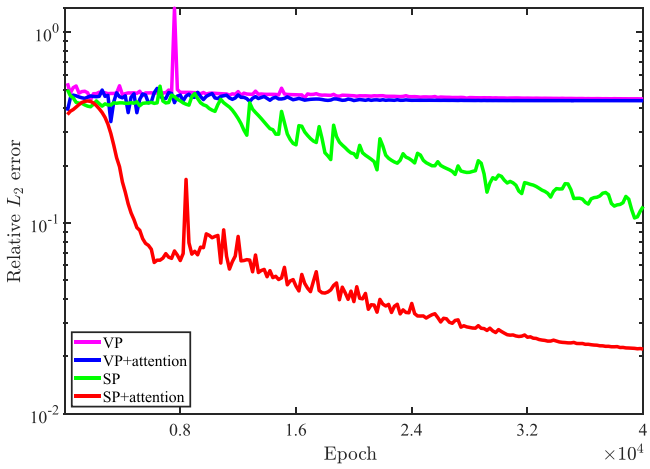


FIG. 7. Lid-driven cavity flow with Reynolds number Re = 100: Variation of relative L_2 errors in velocity predictions by using different neural network architectures and PINNs in velocity–pressure (VP) and streamfunction–pressure (SP) forms. The suffix attention indicates the neural network architecture with attention mechanisms; otherwise, the conventional fully-connected architecture is used.

In addition to using PINNs in the velocity–pressure (VP) form, we also explore PINNs in the streamfunction–pressure (SP) form. For PINNs in the SP form, instead of directly predicting the velocity components (u, v), the feed-forward part of the PINN maps the spatial coordinates to a scalar latent function ψ, along with the pressure p, i.e., $(x, y) \rightarrow (\psi, p)$. The velocity components are then computed by $u = \partial\psi/\partial y$ and $v = -\partial\psi/\partial x$. This formulation ensures the automatic satisfaction of the conservation of mass equation, given by $\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = \frac{\partial^2 \psi}{\partial y \partial x} - \frac{\partial^2 \psi}{\partial y \partial x} = 0$. In our study, we consider two types of network architectures for PINNs in both the VP form and the SP form: the architecture with attention mechanisms and the conventional fully-connected architecture.

A summary of the experimental results is presented in Figs. 7–10 for Re = 100 and Figs. 11 and 12 for Re = 400. Figure 7 illustrates the variation of the relative L_2 errors of the magnitude of the predicted solution $|\boldsymbol{u}(\boldsymbol{x})| = \sqrt{u^2(\boldsymbol{x}) + v^2(\boldsymbol{x})}$ obtained using different strategies with PINNs. Specifically, the VP form and the SP form are considered, utilizing either the architecture with attention mechanisms or the conventional fully-connected architecture. Observing the results, we find that PINNs in the VP form fail to provide accurate approximations, with the relative L_2 errors stagnating during the training process. On the other hand, PINNs in the SP form with the conventional fully-connected architecture achieve slightly higher accuracy but still fall short of satisfactory predictions. However, by employing PINNs in the SP form with attention mechanisms, we achieve the highest accuracy, with the errors reaching approximately 2×10^{-2}. Figure 8 displays the corresponding predicted velocity components in the x- and y-directions. Additionally, Fig. 9 illustrates a comparison between the predicted and reference flow fields, accompanied by the corresponding absolute errors. These predictions exhibit good agreement with the reference solutions provided in Ref. 38. Furthermore, for the simulation of the lid-driven cavity flow, the training process of the PINN models is highly sensitive to the learning rate. This sensitivity is evident in Fig. 10. Fixed learning rate schedules tend to cause the neural network to become trapped in local minima. However, utilizing an exponentially decreasing learning rate schedule helps mitigate this issue, resulting in a more stable training process.

We conduct simulations of the lid-driven cavity flow at a larger Reynolds number (Re = 400), as depicted in Figs. 11 and 12. Our initial attempts to train PINN models from scratch prove inadequate in accurately approximating the solutions. To overcome this limitation, we employ a transfer learning strategy. Specifically, instead of training an entirely new network, we utilize a pre-trained network that had been trained to solve the equation at Re = 100 and retrain it to fit the equation at Re = 400. To evaluate the effectiveness of the transfer learning strategy, we compare its performance with that of training a new network from scratch. For fair comparison, we ensure that the total number of training points, the total number of training iterations, and the network size are the same in both cases. In the transfer learning strategy, the network is initially pre-trained for 40 000 Adam iterations to solve the equation with Re = 100 and then retrained for another 40 000 Adam iterations to fit the equation with Re = 400. In the case of the strategy without transfer learning, the network is trained from scratch using 80 000 Adam iterations. Figure 11 presents the predicted velocity slices at locations $x = y = 0.5$, comparing the results obtained with and without transfer learning. Furthermore, Fig. 12 presents a visual comparison between the predicted flow fields obtained with and without transfer learning, alongside the reference flow fields. Remarkably, the PINNs trained using the transfer learning strategy significantly improve accuracy compared to those trained from scratch. We postulate that this improvement is attributed to the transfer learning strategy's ability to leverage prior information obtained from the simulation of Re = 100. By incorporating this prior knowledge, the transfer learning strategy not only enhances solution accuracy but also has the potential to expedite the training process. For example, instead of solving the equations anew when the Reynolds number increases, the pre-trained PINN model for a lower Reynolds

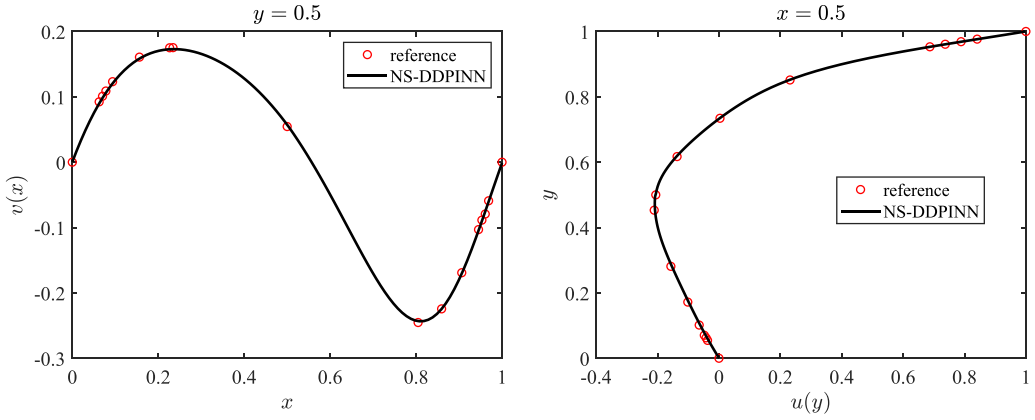


FIG. 8. Lid-driven cavity flow with Reynolds number Re = 100: Comparison between the predicted velocity slices using PINNs in the SP form with attention mechanisms and the reference solutions, at locations $x = y = 0.5$.

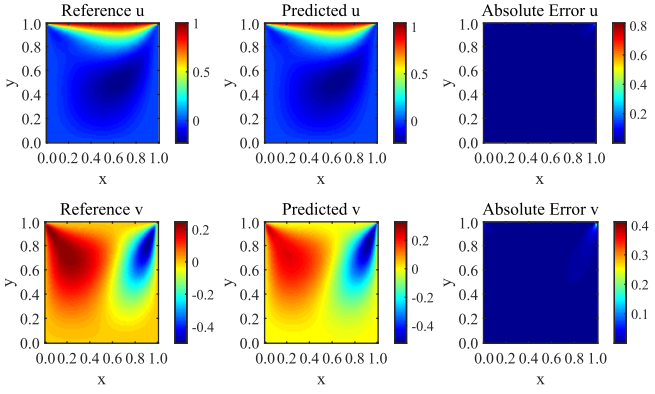


FIG. 9. Lid-driven cavity flow with Reynolds number Re = 100: Comparison between the reference solutions and the predicted solutions using PINNs in the SP form with attention mechanisms.

number can be transferred to the model for a larger Reynolds number, requiring only a few additional iterations to achieve accurate solutions.

D. Inverse problem: 2D cylinder wake

We investigate the inverse problem related to the 2D cylinder wake flow, known for its complex dynamic behavior and transition phenomena at different Reynolds numbers. Specifically, we focus on the case of Reynolds number Re = 100, where the parameters are set to $\lambda_1 = 1$ and $\lambda_2 = 0.01$. The computational domain Ω is defined by the spatial coordinates $[1, 8] \times [-2, 2]$, and the temporal interval is $[0, 20]$. As a reference solution, we utilize high-resolution data obtained from Ref. 1. This reference solution consists of $100 \times 50 \times 200$ regularly sampled points in the x-, y-, and t-coordinates, respectively, covering the entire spatiotemporal domain. We use scattered data on the velocity field (u, v) sampled from this reference data as our training data.

The computational domain Ω is decomposed into four subdomains by evenly dividing the x- and y-coordinates into two parts.

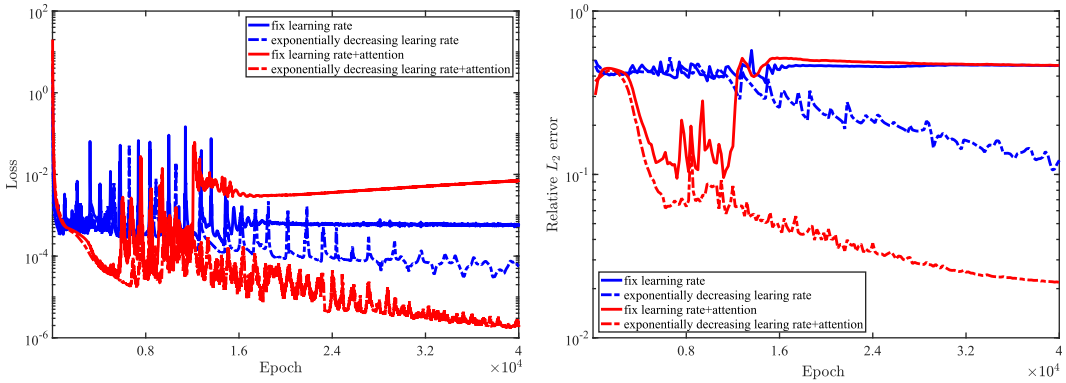


FIG. 10. Lid-driven cavity flow with Reynolds number Re = 100: Variation of training loss and relative L_2 errors of velocity solutions predicted by PINNs in the SP form with different learning rate strategies: fixed learning rate schedule and exponentially decreasing learning rate schedule. The suffix "attention" denotes the neural network architecture with attention mechanisms; otherwise, the conventional fully-connected architecture is used.

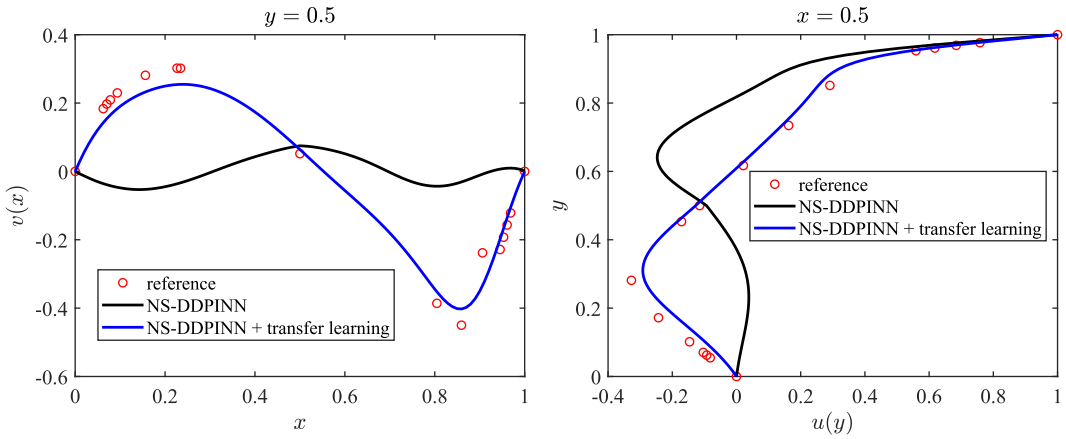


FIG. 11. Lid-driven cavity flow with Reynolds number Re = 400: Comparison of predicted velocity slices obtained from domain decomposition PINNs using two training strategies: training a new network from scratch and transfer learning. The results are compared against the reference solutions at locations $x = y = 0.5$.

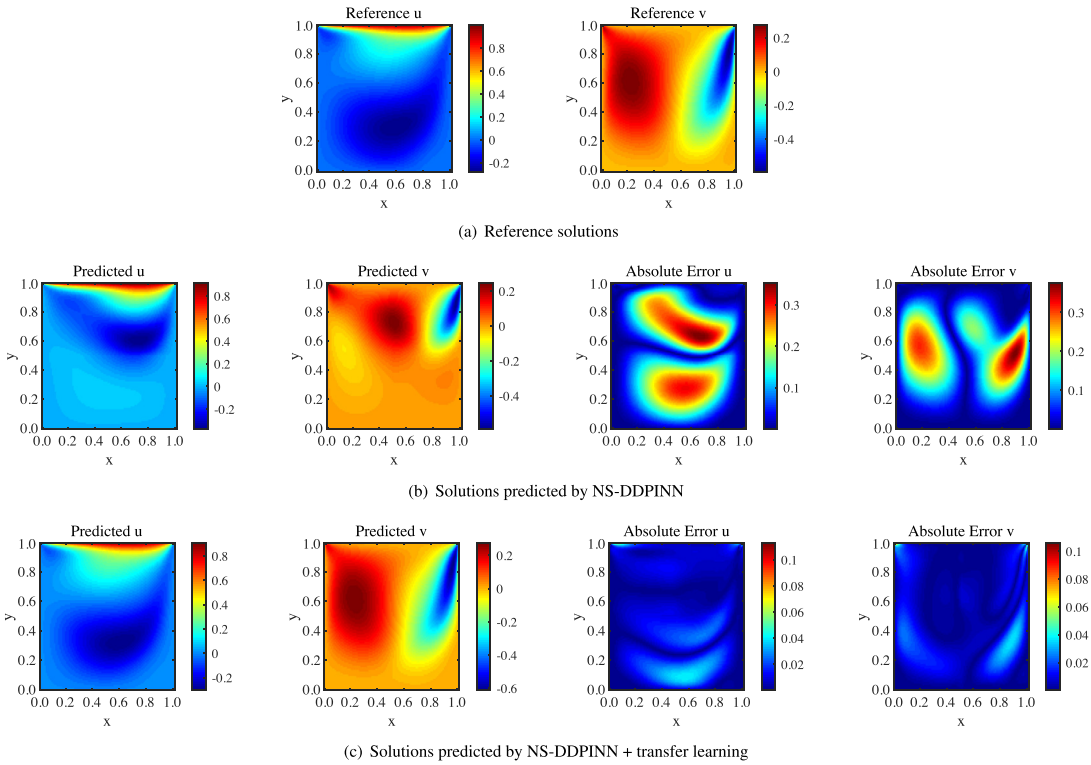


FIG. 12. Lid-driven cavity flow with Reynolds number Re = 400: Comparison between reference solution (a) and predicted solution obtained from domain decomposition PINNs using two training strategies: training a new network from scratch (b) and transfer learning (c).

In the absence of specific instructions, we adhere to the following experimental settings. For the observed data, we randomly sample 2000 points within each spatiotemporal subdomain, corresponding to approximately 0.8% of the total available data. Thus, we have $N_d^u = N_d^{\text{re}} = 2000$ for each subdomain $d \in \{1, \ldots, 4\}$. Regarding the interface points, we utilize 50 points on each boundary and 200 regularly spaced time points along the temporal coordinate. This leads to a total of $N_d^{\text{int}} = 2 \times 50 \times 200$ interface points for each subdomain $d \in \{1, \ldots, 4\}$. In terms of the neural network architecture, we employ a size of 8×20 for all the subdomains. The flux continuity conditions are utilized as additional interface conditions to enforce consistency across subdomains. During training, we conduct 200 000 Adam iterations with a learning rate of 10^{-3}. Additionally, we employ the L-BFGS-B optimization algorithm to further refine the networks.

TABLE VII. Inverse problem for the 2D cylinder wake: (a) Relative L_2 errors of the entire velocity and pressure solutions and (b) the predicted parameters λ_1 and λ_2, predicted by PINNs with domain decomposition. Both the case of clean training data and training data corrupted with 1% uncorrelated Gaussian noise are considered. The effectiveness of two strategies: the dynamic weight strategy and the neural network architecture with attention mechanisms are evaluated by comparing against the original setting.

(a) Relative L_2 errors

		$\varepsilon_u(\%)$	$\varepsilon_v(\%)$	$\varepsilon_p(\%)$
Clean data	Original setting	0.8071	2.4566	11.6399
	Dynamic weight	0.6092	1.9801	6.4670
	Attention mechanism	0.6127	1.9800	11.4269
1% noise	Original setting	0.7592	2.3128	22.7733
	Dynamic weight	0.5892	1.9563	3.7268
	Attention mechanism	0.6078	1.8188	4.5084

(b) Identified parameters

		λ_1	λ_2
Clean data	Original settings	(0.9981,0.9981,0.9982,0.9981)	(0.0111,0.0111,0.0109,0.0109)
	Dynamic weights	(0.9989,0.9989,0.9989,0.9990)	(0.0109,0.0106,0.0108,0.0106)
	Attention mechanism	(0.9993,0.9993,0.9993,0.9993)	(0.0109,0.0107,0.0109,0.0108)
1% noise	Original settings	(0.9976,0.9976,0.9976,0.9977)	(0.0110,0.0107,0.0110,0.0107)
	Dynamic weights	(0.9984,0.9984,0.9984,0.9984)	(0.0109,0.0106,0.0107,0.0106)
	Attention mechanism	(0.9984,0.9985,0.9984,0.9984)	(0.0107,0.0106,0.0107,0.0106)

The results of the inverse problem are summarized in Table VII and Fig. 13. Table VII presents the relative L_2 errors of the entire velocity and pressure solutions, as well as the identified parameters λ_1 and λ_2 obtained using PINNs with domain decomposition. We evaluate the effectiveness of two strategies: the dynamic weight strategy and the neural network architecture with attention mechanisms. These strategies are compared against the original setting to assess their superiority in improving the accuracy. Additionally, to investigate the robustness and generalization capabilities of the PINNs with domain decomposition method for inverse problems, we consider two scenarios: clean training data and training data corrupted with 1% uncorrelated Gaussian noise. Figure 13 illustrates the variations of the identified parameters λ_1 and λ_2 in all four subdomains during the training process. The results shown are obtained using clean training data and the dynamic weight strategy.

Our findings indicate that both the dynamic weight strategy and the neural network architecture with attention mechanisms contribute to improved predictive accuracy. Even when trained with noisy data, the PINNs can accurately identify the unknown parameters λ_1 and λ_2. For instance, when employing the dynamic weight strategy, the errors in estimating λ_1 and λ_2 for clean training data are (0.1128%, 0.1055%, 0.1098%, 0.1047%) and (9.2521%, 6.5269%, 8.1078%, 6.4819%) in the four subdomains, respectively. For noisy training data, the errors are (0.1627%, 0.1578%, 0.1631%, 0.1570%) and (8.6223%, 6.0207%, 7.4044%, 6.01916%), respectively. Furthermore, the PINN models not only yield accurate predictions for the entire velocity fields but also

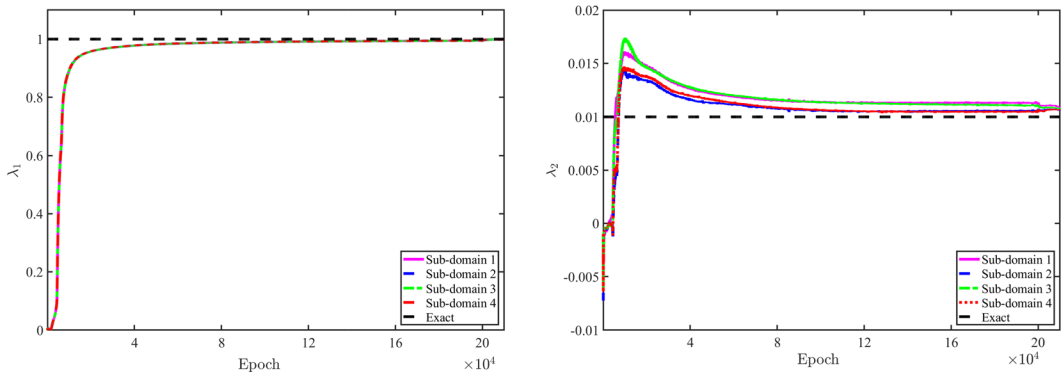
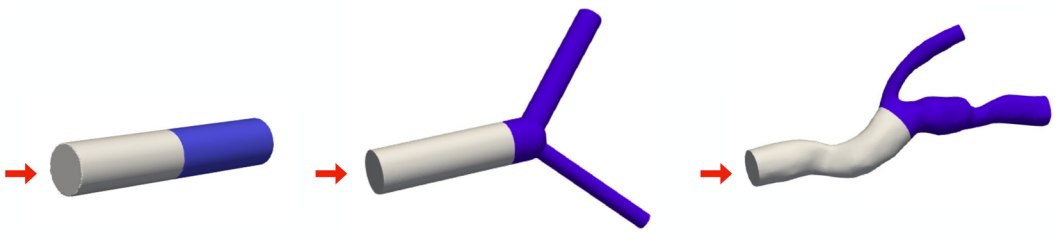
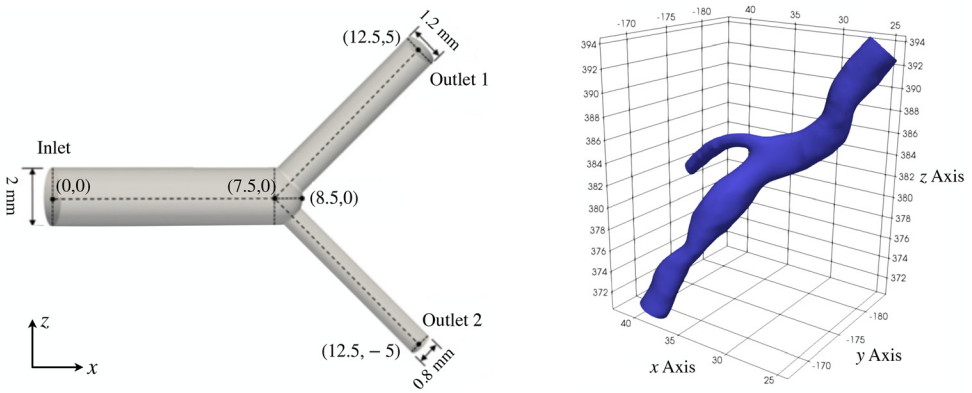


FIG. 13. Inverse problem for the 2D cylinder wake: Variation of the identified parameters λ_1 and λ_2 in all four subdomains during the training process, in which clean training data and dynamic weight strategy are used.



(a) Simulation geometries



(b) The coordinate information of the cylindrical bifurcation and the coronary artery

FIG. 14. 3D Blood Flows: (a) The simulation geometries: (left to right) a cylinder, a cylindrical bifurcation, and a realistic coronary artery. The red arrows represent the direction of the inflow. Different colors in geometries represent that the domains are decomposed into subdomains. (b) The coordinate information of the cylindrical bifurcation (left) and the realistic coronary artery (right).

provide qualitatively reliable estimations for the entire pressure field, even without any training data specifically targeting the pressure.

E. Modeling of 3D blood flows

We utilize the proposed method to simulate the 3D blood flow in both synthetic flow geometries and real blood vessels. Similar to Refs. 39 and 40, we consider stationary systems, where the blood flow is treated as an incompressible Newtonian fluid with a density of $\rho = 1050\,\mathrm{kg\,m^3}$ and a dynamic viscosity of $\mu = 0.003\,85\,\mathrm{Pa\,s}$. Figure 14(a) illustrates three geometries that are considered: a cylinder (with a diameter of 1 mm and a length of 5 mm), a cylindrical bifurcation, and a realistic coronary artery. The cylindrical bifurcation, shown in Fig. 14(b), consists of a parent tube with a diameter of 2 mm, where the left end serves as the inlet and the right end connects to a hemisphere with a radius of 1 mm. Two daughter branches intersect the hemisphere, each with an outlet of diameters 1.2 and 0.8 mm, respectively.

As for the boundary conditions employed in the simulation of the 3D blood flow, we implement the following settings. First, we enforce a no-slip condition on the geometry walls, which means the velocities u, v, and w are all set to zero. At the inlet, a parabolic profile for the velocity is imposed, with the inflow directed perpendicular to the inlet plane. Specifically, the magnitude of the velocity at any point is given by

$$u(r) = u_{\max}\left(1 - \frac{r^2}{R^2}\right),$$

where $u_{\max} = 0.3\,\mathrm{m/s}$ and r represents the distance from the point to the center, while R is the distance from the inlet center to the inlet edge. Thus, the velocity in the inlet plane is maximum at the center and decreases to zero toward the edges. For the outlet boundary condition, zero pressure conditions are used instead of velocity conditions. Additionally, to ensure mass flow continuity, the difference between the mass flow through the inlet plane and the mass flow through the outlet plane is enforced in the loss function during the optimization process.

The remaining experimental settings are as follows. As shown in Fig. 14(a), each geometry is decomposed into two subdomains, with different colors representing different subdomains. Referring to the subdomain containing the inlet plane as the first subdomain and the remaining subdomain as the second subdomain, the interface between them corresponds to the outlet plane of the first subdomain and the inlet plane of the second subdomain. The exchange of interface information across neighboring subdomains is conducted as follows: (1) The velocity on the interface plane, predicted by the PINN model associated with the first subdomain, is employed as the inlet conditions for the second subdomain; (2) The pressure on the interface plane, predicted by the PINN model of the second subdomain, serves as the outlet conditions for the first subdomain. Additionally, for each

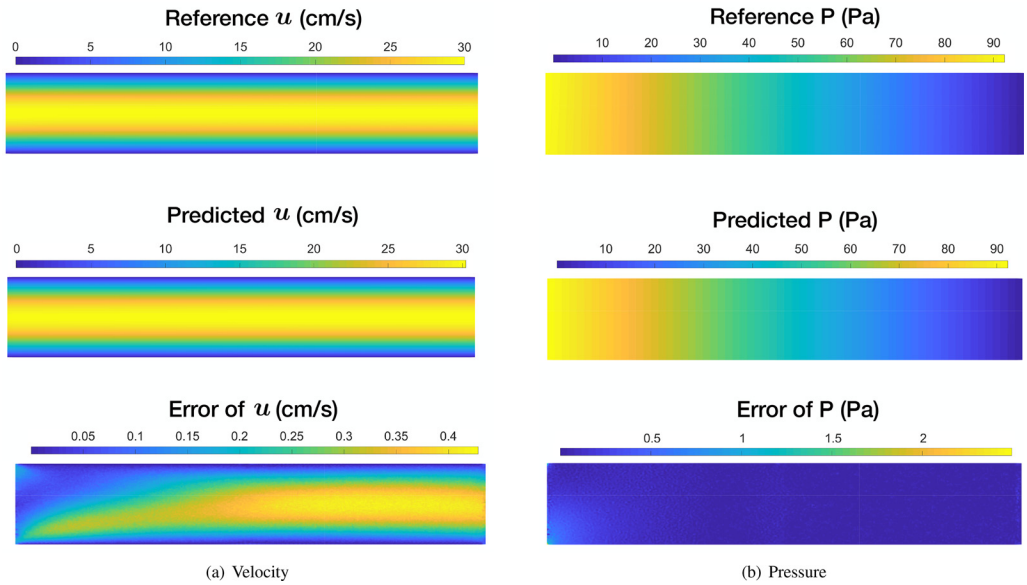
TABLE VIII. 3D blood flows: relative L_2 errors of the predicted velocity and pressure of subdomains for various simulation geometries.

	$\varepsilon_{\|\boldsymbol{u}\|}$ (%)		ε_p (%)	
	Domain first	Domain second	Domain first	Domain second
Cylinder	2.0638	2.1603	5.0621	2.9897
Bifurcation	7.7337	7.9265	1.7239	6.0803
Coronary artery	6.4113	8.3594	0.7169	2.3803

subdomain, the difference between the mass flow through the inlet plane and the mass flow through the outlet plane is enforced in the loss function. During each training iteration, a total of 2000 points are randomly sampled from the inlet, 1000 points from the outlet, 2000 points from the geometry walls, and 3000 interior points (residual points) from the training data. Additionally, 8000 points from each plane are used for calculating the mass flow. For all cases, Adam optimization is employed with an exponentially decreasing learning rate schedule. The initial learning rate is set to $5 \times e^{-4}$. Specifically, for the cylinder case, we perform 200 000 Adam iterations with a learning rate decay factor of 0.9 every 4000 iterations. For the cylindrical bifurcation and coronary artery cases, we conduct 800 000 Adam iterations with a learning rate decay factor of 0.95 every 40 000 iterations. During the inference stage, approximately 1×10^6 points are sampled throughout the domain for each geometry. These points are used to compute the error compared to the reference solutions, which are obtained using the commercial software CFX (ANSYS, Canonsburg, PA).

A summary of the experimental results is provided in Table VIII, along with visual representations in Figs. 15–17. To better highlight the disparity between the predicted and reference values, we opt to present velocity in centimeters per second (cm/s) rather than meters per second (m/s). Table VIII presents the relative L_2 errors of the predicted velocity and pressure for the subdomains in various simulation geometries. The magnitude of the velocity solution, denoted as $|\boldsymbol{u}(\boldsymbol{x})| = \sqrt{u^2(\boldsymbol{x}) + v^2(\boldsymbol{x}) + w^2(\boldsymbol{x})}$, is considered for velocity comparisons. Across all three geometries, the relative L_2 errors for all subdomains reach the order of 10^{-2}. Figure 15, 16, and 17 provide visual snapshots of the velocity and pressure fields, as well as the absolute errors, at selected faces for the cylinder, cylindrical bifurcation, and coronary artery, respectively. In the case of the cylinder, as depicted in Fig. 15, the predicted solution closely matches the reference solution. The deviation between the maximum absolute error of velocity (or pressure) and the maximum velocity (or pressure) value of 30 cm/s (or 92.4 Pa) from the reference is less than 2% (or 3%). For the cylindrical bifurcation, Fig. 16 demonstrates that although there are slight deviations in the branching area, the predicted solution is generally in good agreement with the reference solution in most regions. Regarding the coronary artery, the overall distribution of the predicted solution, illustrated in Figs. 17(b)–17(d), aligns well with the reference solution, with minor differences observed in a few areas.

It is noteworthy that the PINN method's simulation time significantly exceeded that of the commercial software CFX. Specifically, when using CFX, the simulation for the 3D blood flow in all the three geometries finished within a few minutes. In contrast, applying the proposed domain decomposition PINN method prolonged the simulation time to 13.6 h for the cylinder case and a substantial 84.5 h for both the cylindrical bifurcation and the realistic coronary artery cases. In fact, it is widely acknowledged that the current generation of PINNs is not as accurate or efficient at solving standard forward problems as traditional CFD methods.[41] This limitation is attributed to the optimization of PINN models, characterized by a high-dimensional non-convex nature, a challenge inherent in all neural networks. Consequently,



(a) Velocity (b) Pressure

FIG. 15. Cylinder: Comparison of the reference and predicted solutions at the face $y = 0$, showcasing velocity (a) and pressure (b) fields, accompanied by the corresponding absolute errors.

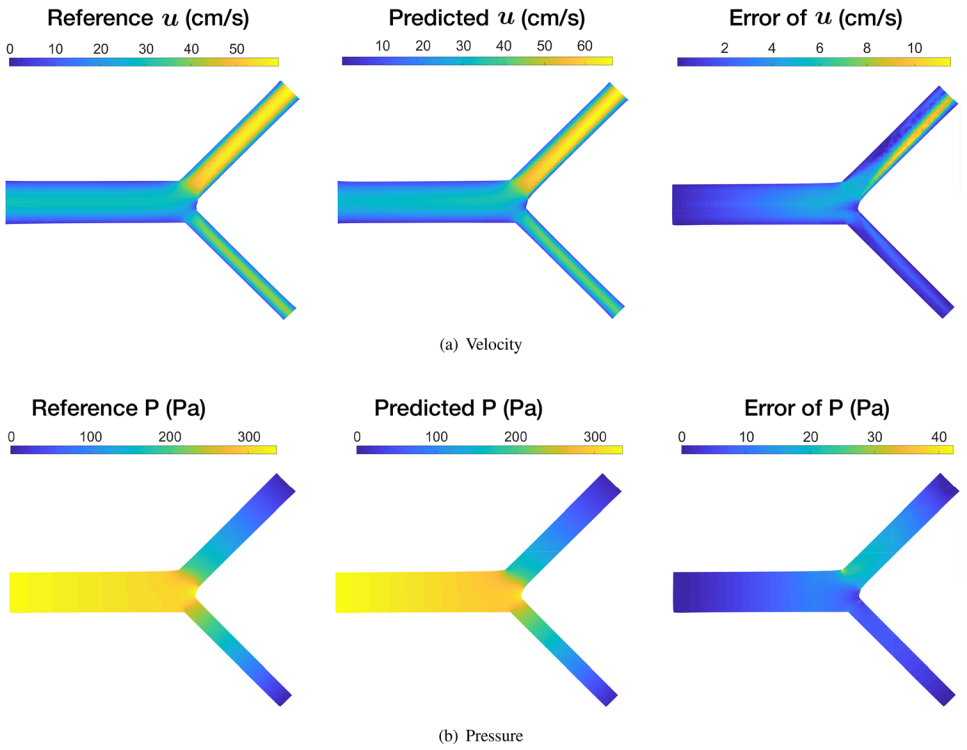


FIG. 16. Bifurcation: Comparison of the reference and predicted solutions at the face $y = 0$, showcasing velocity (a) and pressure (b) fields, accompanied by the corresponding absolute errors.

the training process for the PINN models required a significant number of iterations to achieve convergence. To provide context, the cylinder case involved 200 000 iterations, while both the cylindrical bifurcation and the realistic coronary artery cases necessitated 800 000 iterations each.

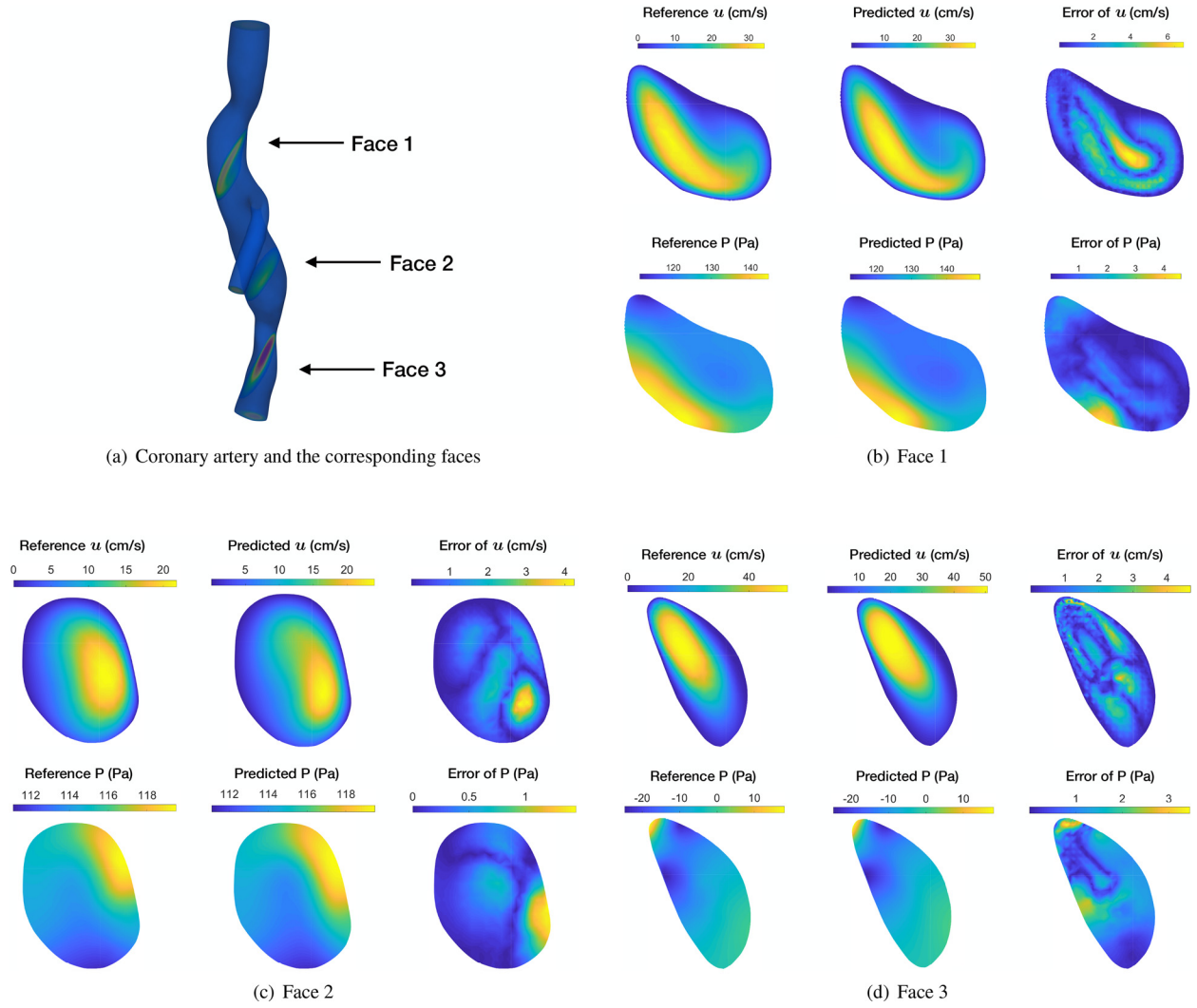
Nevertheless, the PINN model possesses a distinct advantage: even when trained on sparse or noisy observed data, it showcases the ability to predict dense velocity fields. This capability is exemplified by the 2D cylinder flow problem at $\mathrm{Re} = 100$, illustrating the model's robust performance under challenging conditions.

V. SUMMARY AND DISCUSSION

In this study, we delve into the application of the physics-informed neural network (PINN) method with domain decomposition for solving incompressible Navier–Stokes equations. The domain decomposition PINN method employs individual PINNs for each subdomain to compute local solutions, which are connected through additional continuity conditions at the interfaces. Our evaluation focuses on the predicted accuracy, convergence, and the impact of different strategies on the performance. Specifically, we explore various continuity conditions at the interfaces to enhance interface continuity and improve predictive accuracy. To address gradient pathologies commonly encountered in PINN methods, we introduce two approaches: the dynamic weight method and a novel neural network architecture incorporating attention mechanisms.

To demonstrate the effectiveness of the proposed method, we apply it to a diverse set of forward and inverse problems involving incompressible Navier–Stokes flow scenarios. This includes solving benchmark problems such as the 2D Kovasznay flow, the 3D Beltrami flow, the 2D lid-driven cavity flow, and the 2D cylinder wake. Additionally, we conduct 3D blood flow simulations for synthetic and real blood vessels. Our experimental results underscore the capability and the versatility of the domain decomposition PINN method in accurately solving incompressible Navier–Stokes flow problems. Notably, we observe that the continuity conditions at the interfaces significantly enhance interface continuity and predictive accuracy. Furthermore, the dynamic weight strategy and attention mechanisms in the network architecture contribute to the improved predictive accuracy. These methods promote a more consistent behavior of gradients, enhancing the overall balance, stability, and convergence speed.

The domain decomposition PINN method allows straightforward application to parallel computing, which can improve the computational efficiency. However, it is important to note that, in this study, we do not delve into the specific implementation of this method on distributed systems or the parallel efficiency of the method. Exploring the perspectives for parallel computation of the method in future work would be of great interest. Furthermore, in future research, we aim to extend domain decomposition to larger and more complex computational domains for Navier–Stokes equations. This would involve exploring methods to speed up the information interaction between



(a) Coronary artery and the corresponding faces

(b) Face 1

(c) Face 2

(d) Face 3

FIG. 17. Coronary artery: (a) Geometry of the simulated coronary artery and its three faces. (b)–(d) Comparison of the reference and predicted solutions at three faces, showcasing velocity and pressure fields, accompanied by the corresponding absolute errors.

subdomains and maintain or even improve accuracy when the number of subdomains is large. Additionally, we recognize the potential of adopting a properly set configuration in the corresponding subdomain, including the network architecture, the optimization method, and training points. Although not covered in this paper, this flexibility opens up the possibility of formulating prior knowledge and utilizing a meta-learning algorithm to optimize the hyper-parameters. Such advancements would further enhance the performance and adaptability of the domain decomposition PINN method. Furthermore, the PINN model exclusively learns the solution function for a specific instance of a PDE and lacks the capacity to generalize to other instances without re-optimization. For example, when a PINN model is trained on observed data at a specific Reynolds number, it may not accurately predict flows at different Reynolds numbers unless the model undergoes re-optimization. In contrast, the concurrent work related to deep operator network (DeepONet)[42] has demonstrated the ability to overcome this limitation by learning across multiple PDE instances. This presents an intriguing avenue for future research, and investigating the integration of such advancements could enhance the performance of the domain decomposition PINN method.

ACKNOWLEDGMENTS

This work was partially supported by the National Natural Science Foundation of China (NSFC) (Nos. 62161160312, 12071461, 12101589, 12101588, 12001520, and 12371436), the Shenzhen Fundamental Research Program (No. RCYX20200714114735074), and the Changsha Science and Technology Bureau grant (No. KH2301001).

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Linyan Gu: Methodology (equal); Software (equal); Writing – original draft (equal). **Shanlin Qin:** Software (equal); Validation (equal); Visualization (equal). **Lei Xu:** Methodology (equal); Software (equal). **Rongliang Chen:** Conceptualization (equal); Methodology (equal); Supervision (equal); Writing – review & editing (equal).

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request.

REFERENCES

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

[2]D. C. Psichogios and L. H. Ungar, "A hybrid neural network-first principles approach to process modeling," AIChE J. **38**, 1499–1511 (1992).

[3]I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," IEEE Trans. Neural Networks **9**, 987–1000 (1998).

[4]J. Hou, Y. Li, and S. Ying, "Enhancing PINNS for solving PDES via adaptive collocation point movement and adaptive loss weighting," Nonlinear Dyn. **111**, 15233–15261 (2023).

[5]G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," Nat. Rev. Phys. **3**, 422–440 (2021).

[6]M. Raissi, A. Yazdani, and G. E. Karniadakis, "Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations," Science **367**, 1026–1030 (2020).

[7]Z. Cao, K. Liu, K. Luo, Y. Cheng, and J. Fan, "Efficient optimization design of flue deflectors through parametric surrogate modeling with physics-informed neural networks," Phys. Fluids **35**, 125149 (2023).

[8]S. K. Biswas and N. Anand, "Three-dimensional laminar flow using physics informed deep neural networks," Phys. Fluids **35**, 121703 (2023).

[9]S. El-Sapa, "Time-periodic electroosmotic analysis of couple stress fluid in nanofluidic channels with slippages," Chin. J. Phys. **87**, 330–353 (2024).

[10]S. El-Sapa and A. Al-Hanaya, "Effects of slippage and permeability of couple stress fluid squeezed between two concentric rotating spheres," Phys. Fluids **35**, 103112 (2023).

[11]S. El-Sapa, "Cell models for micropolar fluid past a porous micropolar fluid sphere with stress jump condition," Phys. Fluids **34**, 082014 (2022).

[12]X. Jin, S. Cai, H. Li, and G. E. Karniadakis, "NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations," J. Comput. Phys. **426**, 109951 (2021).

[13]S. Goraya, N. Sobh, and A. Masud, "Error estimates and physics informed augmentation of neural networks for thermally coupled incompressible Navier Stokes equations," Comput. Mech. **72**, 267 (2023).

[14]S. Hijazi, M. Freitag, and 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**(1), 1–38 (2023).

[15]N. Geneva and N. Zabaras, "Quantifying model form uncertainty in Reynolds-averaged turbulence models with Bayesian deep neural networks," J. Comput. Phys. **383**, 125–147 (2019).

[16]N. Wandel, M. Weinmann, M. Neidlin, and R. Klein, "Spline-PINN: Approaching PDEs without data using fast, physics-informed hermite-spline CNNs," in *Proceedings of the AAAI Conference on Artificial Intelligence* (AAAI, 2022), Vol. 36, pp. 8529–8538.

[17]J. Oldenburg, F. Borowski, A. Öner, K.-P. Schmitz, and M. Stiehm, "Geometry aware physics informed neural network surrogate for solving Navier-Stokes equation (GAPINN)," Adv. Model. Simul. Eng. Sci. **9**, 8 (2022).

[18]H. Eivazi, M. Tahani, P. Schlatter, and R. Vinuesa, "Physics-informed neural networks for solving Reynolds-averaged Navier-Stokes equations," Phys. Fluids **34**, 075117 (2022).

[19]J. Sirignano, J. F. MacArt, and J. B. Freund, "DPM: A deep learning PDE augmentation method with application to large-eddy simulation," J. Comput. Phys. **423**, 109811 (2020).

[20]L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "DeepXDE: A deep learning library for solving differential equations," SIAM Rev. **63**, 208–228 (2021).

[21]Y. Gu, H. Yang, and C. Zhou, "Selectnet: Self-paced learning for high-dimensional partial differential equations," J. Comput. Phys. **441**, 110444 (2021).

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

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

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

[25]S. Wang, X. Yu, and P. Perdikaris, "When and why PINNs fail to train: A neural tangent kernel perspective," J. Comput. Phys. **449**, 110768 (2022).

[26]X.-C. Cai and Y. Saad, "Overlapping domain decomposition algorithms for general sparse matrices," Numer. Linear Algebra Appl. **3**, 221–237 (1996).

[27]E. Kharazmi, Z. Zhang, and G. E. Karniadakis, "hp-VPINNs: Variational physics-informed neural networks with domain decomposition," Comput. Methods Appl. Mech. Eng. **374**, 113547 (2021).

[28]A. D. Jagtap and G. E. Karniadakis, "Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations," Commun. Comput. Phys. **28**, 2002–2041 (2020).

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

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

[31]W. Li, X. Xiang, and Y. Xu, "Deep domain decomposition method: Elliptic problems," in *Mathematical and Scientific Machine Learning* (PMLR, 2020), pp. 269–286.

[32]V. Dwivedi, N. Parashar, and B. Srinivasan, "Distributed learning machines for solving forward and inverse problems in partial differential equations," Neurocomputing **420**, 299–316 (2021).

[33]B. Moseley, A. Markham, and T. Nissen-Meyer, "Finite basis physics-informed neural networks (FBPINNs): A scalable domain decomposition approach for solving differential equations," arXiv:2107.07871 (2021).

[34]K. Li, K. Tang, T. Wu, and Q. Liao, "D3M: A deep domain decomposition method for partial differential equations," IEEE Access **8**, 5283–5294 (2020).

[35]A. Heinlein, A. Klawonn, M. Lanser, and J. Weber, "Combining machine learning and domain decomposition methods for the solution of partial differential equations: A review," GAMM-Mitt. **44**, e202100001 (2021).

[36]J. Yu, L. Lu, X. Meng, and G. E. Karniadakis, "Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems," Comput. Methods Appl. Mech. Eng. **393**, 114823 (2022).

[37]C. R. Ethier and D. A. Steinman, "Exact fully 3D Navier-Stokes solutions for benchmarking," Numer. Methods Fluids **19**, 369–375 (1994).

[38]U. Ghia, K. N. Ghia, and C. Shin, "High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method," J. Comput. Phys. **48**, 387–411 (1982).

[39]W. Fenz, J. Dirnberger, and I. Georgiev, "Blood flow simulations with application to cerebral aneurysms," in *Proceedings of the Modeling and Simulation in Medicine Symposium (MSM)* (Society for Computer Simulation International, San Diego, CA, 2016), pp. 1–8.

[40]P. Moser, W. Fenz, S. Thumfart, I. Ganitzer, and M. Giretzlehner, "Modeling of 3D blood flows with physics-informed neural networks: Comparison of network architectures," Fluids **8**, 46 (2023).

[41]S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis, "Physics-informed neural networks (PINNs) for fluid mechanics: A review," Acta Mech. Sin. **37**, 1727–1738 (2021).

[42]L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators," Nat. Mach. Intell. **3**, 218–229 (2021).