Hybrid deep learning and iterative methods for accelerated solutions of viscous incompressible flow

Heming Bai[a], Xin Bian[a,*]

[a]*State Key Laboratory of Fluid Power and Mechatronic Systems, Department of Engineering Mechanics, Zhejiang University, Hangzhou, 310027, China*

Abstract

The pressure Poisson equation, central to the fractional step method in incompressible flow simulations, incurs high computational costs due to the iterative solution of large-scale linear systems. To address this challenge, we introduce HyDEA (Hybrid Deep lEarning line-search directions and iterative methods for Accelerated solutions), a novel framework that synergizes deep learning with classical iterative solvers. It leverages the complementary strengths of a deep operator network (DeepONet) – capable of capturing large-scale features of the solution – and the conjugate gradient (CG) or a preconditioned conjugate gradient (PCG) (with Incomplete Cholesky, Jacobi, or Multigrid preconditioner) method, which efficiently resolves fine-scale errors. Specifically, within the framework of line-search methods, the DeepONet predicts search directions to accelerate convergence in solving sparse, symmetric-positive-definite linear systems, while the CG/PCG method ensures robustness through iterative refinement. The framework seamlessly extends to flows over solid structures via the decoupled immersed boundary projection method, preserving the linear system's structure. Crucially, the DeepONet is trained on *fabricated* linear systems rather than flow-specific data, endowing it with inherent generalization across geometric complexities and Reynolds numbers without retraining. Benchmarks demonstrate HyDEA's superior efficiency and accuracy over the CG/PCG methods for flows with no obstacles, single/multiple stationary obstacles, and one moving obstacle – using *fixed network weights*. Remarkably, HyDEA also exhibits super-resolution capability: although the DeepONet is trained on a 128×128 grid for Reynolds number $Re = 1000$, the hybrid solver delivers accurate solutions on a 512×512 grid for $Re = 10,000$ via interpolation, despite discretizations mismatch. In contrast, a purely data-driven DeepONet fails for complex flows, underscoring the necessity of hybridizing deep learning with iterative methods. HyDEA's robustness, efficiency, and generalization across geometries, resolutions, and Reynolds numbers highlight its potential as a transformative solver for real-world fluid dynamics problems.

Keywords: Hybrid method, Deep learning line-search method, Incompressible flows, Immersed boundary method, Pressure Poisson equation
PACS: 0000, 1111
2000 MSC: 0000, 1111

1. Introduction

Computational fluid dynamics (CFD) has revolutionized our understanding of fluid mechanics and enabled innovative real-world applications through advanced numerical methods [1]. However, solving the Navier-Stokes (NS) equations remains computationally expensive, especially for unsteady or geometrically complex flows that require substantial resources. This challenge is compounded by the necessity of performing separate simulations for each flow condition, rendering conventional CFD approaches inefficient for engineering applications like active flow control [2] and structural shape optimization [3]. These limitations underscore the pressing need for novel techniques to accelerate simulations and minimize redundant computations across similar flow scenarios.

Deep learning has demonstrated remarkable success in fields such as computer vision [4] and natural language processing [5], and its strong predictive capabilities have recently extended to solving partial differential equations (PDEs). Physics-informed neural networks (PINNs) [6, 7], for instance, embed PDE constraints directly into training via automatic differentiation, enabling solutions without relying solely on large datasets. Despite their promise, PINNs still face challenges in computational efficiency and accuracy in fluid dynamics [8, 9, 10], calling for further methodological advances [11, 12].

Alternatively, data-driven surrogate models trained on high-fidelity data offer fast solutions to NS equations [13, 14], bypassing costly simulations during deployment. However, these models often generalize poorly beyond their training distribution. Recent efforts to address this fall into two categories:
1. Novel architectures: Operator learning frameworks [15], such as Deep operator network (DeepONet) [16] and Fourier

*Corresponding author
Email address: bianx@zju.edu.cn (Xin Bian)

neural operator [17], approximate PDE solutions by encoding spatio-temporal dynamics. They have been applied to Burgers' equation [16], multiphase flows [18, 19], steady compressible flows [20], and unsteady incompressible flows [21, 22]. Yet, their performance remains tied to the distribution of the operator parameters sampled in the training dataset.
2. Training strategies: Via pre-training and fine-tuning operations, meta-learning [23], self-supervised learning [24], and transfer learning [25] improve adaptability to new flow conditions. However, they usually require supplementary data, which may be unavailable in practical applications.

The critical limitation of data-driven models is their inability to generalize to out-of-training distribution, such as varying Reynolds numbers or modified boundary conditions that alter the underlying dynamics. This recognition has spurred the development of *hybrid methods*, which integrate deep learning with classical numerical methods rather than outright eliminating the latter. For example, neural networks [26, 27, 28, 29] have been used to accelerate solution to the pressure Poisson equation (PPE) in the fractional step methods[30, 31], a major computational bottleneck in incompressible NS simulations. However, these models are employed as end-to-end mapping for one shot at each instance and do not improve its own prediction with iterative efforts. More importantly, they often rely on flow-specific data, leading to performance degradation for out-of-distribution scenarios, a limitation shared by purely data-driven models. Kaneda et al. [32] proposed the deep conjugate direction method (DCDM), which trains on the PPE coefficient matrix rather than flow data, leading to superior generalization. While promising, DCDM has only been tested on Euler equations, leaving its efficacy for NS equations unexplored. Additionally, it suffers from boundary inaccuracies due to its voxel-based fluid-solid distinction.

Another fundamental challenge is *spectral bias* [33]: neural networks excel at capturing low-frequency features but struggle with high-frequency details. Whereas classical iterative methods, including standard relaxation methods and Krylov subspace techniques, efficiently attenuate high-frequency error components but converge slowly for low-frequency modes without heavily employing advanced techniques such as multigrid or proper preconditioning [34, 35]. These complementary behaviors have inspired a new paradigm of hybrid methods that combine data-driven models with numerical methods to exploit their respective strengths. Notable advances include HINTs [36, 37], Multi-scale neural computing [38], DeepONet-preconditioned Krylov methods [39], and PINN-MG [40], but their direct application to nonlinear PDEs such as NS equations remains underexplored. Chen et al. [41] introduced a wavelet-based neural multigrid method for periodic flows. While the construction of training dataset is innovative, the mapping from source terms to pressure fields based on neural network remains susceptible to distribution shifts, such as unseen obstacles. Notably, the DCDM although effective for generalization due to matrix-derived training, depends exclusively on neural networks [32], leaving it vulnerable to spectral bias, which may reduce robustness and accuracy in complex flows.

Despite progress, no hybrid framework has successfully unified data-driven efficiency with numerical robustness while remaining independent of flow-specific training data. Addressing this gap could unlock scalable, high-fidelity solvers for fluid dynamics and beyond. To tackle this challenge, we propose HyDEA (Hybrid Deep lEarning line-search directions and iterative methods for Accelerated solutions) – a novel framework for solving the PPE in incompressible flow simulations. Based on a finite difference discretization, the PPE amounts to a large, sparse, symmetric, positive-definite linear system of equations. The solution to these equations is formulated as an extremum-seeking problem of a quadratic function for the solution vector. Within the framework of line-search methods [42], a DeepONet is employed to predict new search directions at each iteration, which is in sharp contrast to other classical line-search directions such as those derived from steepest descent (SD) and conjugate gradient (CG) methods [42, 43]. It is termed as Deep learning Line-Search Method or DLSM in brevity, for later reference. Furthermore, HyDEA integrates DLSM with the CG method or one of its preconditioned variants (collectively referred to as CG-type methods), creating a synergistic approach that:

- Preserves generalization: training on fabricated linear systems biased toward the lower spectrum of the PPE matrix, avoiding flow-specific data;
- Integrates DLSM with a CG-type method to balance neural acceleration and numerical stability;
- Incorporates fluid-structure interaction with decoupled immersed boundary projection method (DIBPM) [44], which preserves the linear system's structure for extensive generalization.

Subsequently, HyDEA adopts *identical weights* for neural network and demonstrates through a series of benchmarks that:

- Achieves superior convergence rate compared to the standalone CG-type methods,
- Maintains generalization capability across diverse flow regimes with/without solid structures, and also with super-resolutions.
- Outperforms DCDM and DLSM in both stability and accuracy for complex flows.

By synergistically combing the strengths of neural networks and iterative methods, HyDEA represents a significant step toward scalable, high-fidelity PDE solver capable of handling complex multi-physics simulations.

The remainder of this work is structured as follows: Section 2 presents the numerical foundation, beginning with the fractional step method for incompressible NS equations and DIBPM for solid structures, followed by a detailed exposition of the proposed HyDEA methodology. This section also covers the construction of training dataset, neural network

architecture, and model training protocols. Section 3 demonstrates HyDEA's performance through multiple benchmark cases with comprehensive analysis of convergence acceleration and generalization capability. Finally, Section 4 concludes with key findings and outlines promising directions for future research.

2. Methodology

2.1. Fractional step method for incompressible Navier-Stokes equations

The governing equations are the incompressible NS equations expressed in non-dimensional form as

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \frac{1}{Re} \nabla^2 \mathbf{u}, \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{2}$$

where $\mathbf{u}$, p and Re represent the velocity vector, pressure and Reynolds number, respectively. The numerical solution of the NS equations is challenging, as the pressure term is coupled to the velocity and needs to be updated to implicitly satisfy the incompressibility constraint. The fractional step method is a classical algorithm, which effectively decouples the velocity and pressure. Following Ref. [31], the NS equations can be discretized and derived into a system of algebraic equations as

$$\begin{bmatrix} A & G \\ D & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}^{n+1} \\ \delta p \end{bmatrix} = \begin{bmatrix} \mathbf{r}^n + bc_1 \\ bc_2 \end{bmatrix}, \tag{3}$$

where A represents the implicit operator matrix for the advection-diffusion component, D is the divergence operator matrix, G is the gradient operator matrix, and $\mathbf{r}^n$ represents the explicit terms of the discrete momentum equation. Additionally, bc_1 and bc_2 are boundary condition vectors for the momentum equation and the incompressibility constraint, respectively. $\mathbf{u}^{n+1}$ is the unknown velocity vector. $\delta p = p^{n+1} - p^n$, and this form can reduce the splitting error of the fractional step method [45]. The structure of A and $\mathbf{r}^n$ critically depends on the specific time advancement scheme and mesh configuration. In this work, the NS equations are discretized by the finite difference method on a staggered-mesh, utilizing the explicit second-order Adams-Bashforth scheme for the convective terms and the implicit Crank-Nicolson method for the diffusion terms.

In view of Ref. [31], the fractional step method can be regarded as a block LU decomposition of Eq. (3),

$$\begin{bmatrix} A & 0 \\ D & -DA^{-1}G \end{bmatrix} \begin{bmatrix} I & A^{-1}G \\ 0 & I \end{bmatrix} \begin{bmatrix} \mathbf{u}^{n+1} \\ \delta p \end{bmatrix} = \begin{bmatrix} \mathbf{r}^n + bc_1 \\ bc_2 \end{bmatrix}. \tag{4}$$

The direct calculation of A^{-1} in Eq. (4) is prohibitively expensive, thus an approximate solution for A^{-1} is employed. The Taylor series expansion is applied to A^{-1} and only its first term is kept, $A^{-1} \approx \Delta t$. Subsequently, Eq. (4) transforms to be

$$\begin{bmatrix} A & 0 \\ D & -\Delta t DG \end{bmatrix} \begin{bmatrix} I & \Delta t G \\ 0 & I \end{bmatrix} \begin{bmatrix} \mathbf{u}^{n+1} \\ \delta p \end{bmatrix} = \begin{bmatrix} A & 0 \\ D & -\Delta t DG \end{bmatrix} \begin{bmatrix} \mathbf{u}^{*} \\ \delta p \end{bmatrix} = \begin{bmatrix} \mathbf{r}^n + bc_1 \\ bc_2 \end{bmatrix}. \tag{5}$$

Eq. (5) is typically rewritten in the form of three sequential steps:

$$A\mathbf{u}^* = \mathbf{r}^n + bc_1, \tag{6}$$

$$\Delta t DG \delta p = D\mathbf{u}^* - bc_2, \tag{7}$$

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \Delta t G \delta p, \tag{8}$$

where $\mathbf{u}^*$ is the intermediate velocity vector. Eq. (7) represents the discrete PPE, where DG is a symmetric-positive-definite matrix. Classical iterative methods, such as CG-type methods [42, 43], can be employed to iteratively solve Eq. (7) for δp, which is the most computationally intensive part of the whole solution process.

Finally, the pressure vector is updated as follow:

$$p^{n+1} = p^n + \delta p. \tag{9}$$

2.2. Decoupled immersed boundary projection method

The immersed boundary method (IBM) is a numerical technique for handling fluid-structure interactions, originally developed by Peskin et al. [46] to simulate blood flow in cardiovascular systems. The method incorporates a forcing term $\mathbf{f}$ into NS equations as

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \frac{1}{Re} \nabla^2 \mathbf{u} + \mathbf{f}, \tag{10}$$

$$\nabla \cdot \mathbf{u} = 0. \tag{11}$$

Taira et al. [47] proposed the immersed boundary projection method (IBPM) based on the fractional step method of Perot et al. [31]. However, this proposition leads the pressure variable to couple with the forcing term, resulting in a substantially high computational cost. To address this issue, Li et al. [44] developed the decoupled IBPM or DIBPM expressed in a two-step block LU decomposition, which effectively decouples the velocity, pressure and forcing term.

In the framework of DIBPM, fluid dynamics is expressed in Eulerian framework with Cartesian coordinates, while the immersed boundary is defined at Lagrangian points $\mathbf{X}(\mathbf{s}, t)$, with $\mathbf{s}$ and t denoting the curvilinear coordinates and time, respectively. The force and velocity at $\mathbf{X}(\mathbf{s}, t)$ are denoted by $\mathbf{F}(\mathbf{s}, t)$ and $\mathbf{U}(\mathbf{s}, t)$. The variables defined at two coordinates impose each other with a delta function as

$$\mathbf{f}(\mathbf{s}, t) = \int_{\Gamma} \mathbf{F}(\mathbf{s}, t)\delta(\mathbf{x} - \mathbf{X}(\mathbf{s}, t))d\mathbf{s}, \tag{12}$$

$$\mathbf{U}(\mathbf{s}, t) = \int_{\Omega} \mathbf{u}(\mathbf{x}, t)\delta(\mathbf{X}(\mathbf{s}, t) - \mathbf{x})d\mathbf{x}, \tag{13}$$

where Γ is fluid-solid boundary in Lagrangian curvilinear coordinate, Ω is fluid domain in Eulerian coordinate. The discretization of Eq. (12) and Eq. (13) are represented by the operator matrices H and E, which correspond to the discrete regularization and interpolation operators, respectively. In this work, the discrete delta function of Roma et al. [48] is utilized. Ultimately, the governing equations are discretized and expressed as a system of algebraic equations as

$$\begin{bmatrix} A & -H & G \\ E & 0 & 0 \\ D & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}^{n+1} \\ \delta\mathbf{F} \\ \delta p \end{bmatrix} = \begin{bmatrix} \mathbf{r}^n + bc_1 \\ \mathbf{U_B} \\ bc_2 \end{bmatrix}, \tag{14}$$

where $\delta\mathbf{F} = \mathbf{F}^{n+1} - \mathbf{F}^n$, and $\mathbf{U_B}$ is the Lagrangian velocity on the immersed boundary.

Expressed in a two-step block LU decomposition by DIBPM, the pressure decoupling process is initially conducted with Eq. (14) being rewritten in the following form,

$$\begin{bmatrix} \bar{A} & \bar{G} \\ \bar{D} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{z}^{n+1} \\ \delta p \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{r}}^n \\ bc_2 \end{bmatrix}. \tag{15}$$

Here $\bar{A} = \begin{bmatrix} A & -H \\ E & 0 \end{bmatrix}$, $\bar{G} = \begin{bmatrix} G & 0 \end{bmatrix}^T$, $\bar{D} = \begin{bmatrix} D & 0 \end{bmatrix}$, $\mathbf{z}^{n+1} = \begin{bmatrix} \mathbf{u}^{n+1} & \delta\mathbf{F} \end{bmatrix}^T$, and $\bar{\mathbf{r}}^n = \begin{bmatrix} \mathbf{r}^n + bc_1 & \mathbf{U_B} \end{bmatrix}^T$. Similar to the fractional step method without solid structure [31], the first block LU decomposition of Eq. (15) is performed,

$$\begin{bmatrix} \bar{A} & 0 \\ \bar{D} & -\Delta t\bar{D}\bar{G} \end{bmatrix} \begin{bmatrix} I & \Delta t\bar{G} \\ 0 & I \end{bmatrix} \begin{bmatrix} \mathbf{z}^{n+1} \\ \delta p \end{bmatrix} = \begin{bmatrix} \bar{A} & 0 \\ \bar{D} & -\Delta t\bar{D}\bar{G} \end{bmatrix} \begin{bmatrix} \mathbf{z}^{*} \\ \delta p \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{r}}^n \\ bc_2 \end{bmatrix}. \tag{16}$$

Eq. (16) can be rewritten in the form of three sequential steps as

$$\bar{A}\mathbf{z}^* = \bar{\mathbf{r}}^n, \tag{17}$$

$$\Delta t\bar{D}\bar{G}\delta p = \bar{D}\mathbf{z}^* - bc_2, \tag{18}$$

$$\mathbf{z}^{n+1} = \mathbf{z}^* - \Delta t\bar{G}\delta p. \tag{19}$$

Subsequently, according to the specific definitions of $\bar{A}$, $\bar{G}$, $\bar{D}$, $\mathbf{z}^{n+1}$ and $\bar{\mathbf{r}}^n$, Eq. (17)-(19) can be further rewritten as

$$\begin{bmatrix} A & -H \\ E & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}^* \\ \delta\mathbf{F} \end{bmatrix} = \begin{bmatrix} \mathbf{r}^n + bc_1 \\ \mathbf{U_B} \end{bmatrix}, \tag{20}$$

$$\Delta tDG\delta p = D\mathbf{u}^* - bc_2, \tag{21}$$

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \Delta tG\delta p. \tag{22}$$

As the intermediate velocity $\mathbf{u}^*$ remains coupled with $\delta\mathbf{F}$, the second LU block decomposition is employed for Eq. (20):

$$\begin{bmatrix} A & 0 \\ E & \Delta tEH \end{bmatrix} \begin{bmatrix} I & -\Delta tH \\ 0 & I \end{bmatrix} \begin{bmatrix} \mathbf{u}^* \\ \delta\mathbf{F} \end{bmatrix} = \begin{bmatrix} \mathbf{r}^n + bc_1 \\ \mathbf{U_B} \end{bmatrix}. \tag{23}$$

Eq. (23) can be rewritten in a similar form as Eq. (17)-(19) as

$$A\mathbf{u}^{**} = \mathbf{r}^n + bc_1, \tag{24}$$

$$\Delta tEH\delta\mathbf{F} = \mathbf{U_B} - E\mathbf{u}^{**}, \tag{25}$$

$$\mathbf{u}^* = \mathbf{u}^{**} + \Delta tH\delta\mathbf{F}, \tag{26}$$

where $\mathbf{u}^{**}$ is the intermediate velocity vector of $\mathbf{u}^*$.

In summary, the complete DIBPM computation process is outlined successively as follows:

$$
\begin{aligned}
A\mathbf{u}^{**} &= \mathbf{r}^n + bc_1, && (27)\\
\Delta t E H \delta \mathbf{F} &= \mathbf{U_B} - E\mathbf{u}^{**}, && (28)\\
\mathbf{u}^{*} &= \mathbf{u}^{**} + \Delta t H \delta \mathbf{F}, && (29)\\
\Delta t D G \delta p &= D\mathbf{u}^{*} - bc_2, && (30)\\
\mathbf{u}^{n+1} &= \mathbf{u}^{*} - \Delta t G \delta p, && (31)\\
p^{n+1} &= p^n + \delta p, && (32)\\
\mathbf{F}^{n+1} &= \mathbf{F}^n + \delta \mathbf{F}. && (33)
\end{aligned}
$$

The fractional step and DIBPM methods employed in this work are based on the open-source CFD solver `PetIBM` [49].

2.3. HyDEA: Hybrid Deep lEarning line-search directions and iterative methods for Accelerated solutions

The PPE in Eq. (7) or (30) can be expressed in a compact form:

$$M\delta p = S, \tag{34}$$

where matrix $M \in \mathbf{R}^{a\times a}$ represents the discrete approximation of the Laplace operator, which is sparse, symmetric and positive-definite. δp is unknown and S is the source term, where $\delta p \in \mathbf{R}^a$ and $S \in \mathbf{R}^a$. The solution to Eq. (34) is equivalent to solving a quadratic vector optimization problem [35, 43],

$$\min_{\delta p\in\mathbf{R}^a} g(\delta p) \stackrel{\text{def}}{=} \frac{1}{2}\delta p^T M \delta p - S^T \delta p + c. \tag{35}$$

Line search methods [42] provide a framework for solving Eq. (35), where solution δp_k at kth iteration is progressively updated through a line-search direction d_k with step length α_k to minimize the objective function $g(\delta p)$:

$$
\begin{aligned}
\alpha_k &= \arg\min_{\alpha} g(\delta p_k + \alpha d_k) = \frac{r_k^T d_k}{d_k^T M d_k}, && (36)\\
\delta p_{k+1} &= \delta p_k + \alpha_k d_k, && (37)
\end{aligned}
$$

where $r_k = S - M\delta p_k$ denotes the iterative residual vector.

A different choice of d_k corresponds to a different variant in the family of line search methods. When d_k is taken as r_k, it is known as steepest descent (SD) method. However, the SD method exhibits suboptimal convergence performance. Furthermore, the conjugate direction (CD) methods can be viewed as an extension of the SD method and they construct a set of M-orthogonal vectors $\{d_0, d_1, \cdots, d_k\}$ ($d_i^T M d_j = 0$ for $i \neq j$) as line-search directions at each iteration through Gram-Schmidt (GS) orthogonalization. The conjugate gradient (CG) method represents a specific implementation of the CD methods, where each residual vector r_k is employed in the GS process.

The convergence performance of a line-search method is fundamentally determined by the quality of the selected line-search directions. Theoretically, if the line-search direction aligned exactly with the current iteration error $e_k = \delta p^{true} - \delta p_k$, the line-search method would converge to the exact solution in a single iteration. However, this theoretical scenario is practically unattainable since e_k is unknown. Recognizing this fundamental limitation, we propose to utilize a deep neural network, namely DeepONet, to predict an error e_k^{NN} as a line-search direction at each iteration. By combining e_k^{NN} with the optimal step length given in Eq. (36), this strategy will prove to achieve rapid convergence within a few number of iterations. We denote this novel methodology as Deep learning Line-Search Method (DLSM) for later reference. More specifically, Eq. (34) is transformed into the residual form at each iteration as

$$Me_k = M\Delta(\delta p) = r_k. \tag{38}$$

DLSM employs a deep neural network to provide the mapping between the r_k and e_k, with the algorithmic implementation formally described in Algorithm 1.[1]

Due to the inherent limitations of neural networks in capturing high-frequency features [33], DLSM is potentially incompetent to eliminate high-frequency errors, thus adversely affect the convergence performance as will be demonstrated in Sections 3.1.3 and 3.4. Inspired by the work of Zhang et al. [36], we propose HyDEA, a hybrid strategy that combines DLSM with iterative methods: the DLSM providing a superior global large-scale line-search directions, while iterative methods resolving small-scale errors. Specifically, we focus on employing CG-type methods as representative iterative

[1] Note that the DCDM of Kaneda et al. [32] represents a special case of DLSM, where the line-search directions are made additionally M-orthogonal by a GS process. We shall demonstrate in Section 3.4 that the GS orthogonalizaiton is unnecessary, as a properly trained neural network already suggests effective line-search directions.

Algorithm 1 DLSM: Deep learning Line-Search Method

$r_0 = S - M\delta p_0$
$k = 0$
while $\|r_k\|_2 > \epsilon$ **do**
  $e_k^{NN} = NN(\frac{r_k}{\|r_k\|_2})$
  $k = k + 1$
  $\alpha_{k-1} = \frac{r_{k-1}^T e_{k-1}^{NN}}{(e_{k-1}^{NN})^T M e_{k-1}^{NN}}$
  $\delta p_k = \delta p_{k-1} + \alpha_{k-1} e_{k-1}^{NN}$
  $r_k = S - M\delta p_k$
end while

methods in the HyDEA framework and meanwhile the performance of standalone CG-type methods shall be considered as the baseline for comparisons.

The workflow of HyDEA is presented in Fig. 1, where it receives an initial solution and its corresponding residual for the current CFD time step. Subsequently, it selects to perform whether the DLSM or a CG-type method for the current iteration. After each iteration, the $L2$-norm of the updated residual is evaluated and if it falls below a predefined threshold, the iterative process terminates. Otherwise, the procedure repeats, progressively refining the solution until the convergence criterion is met.

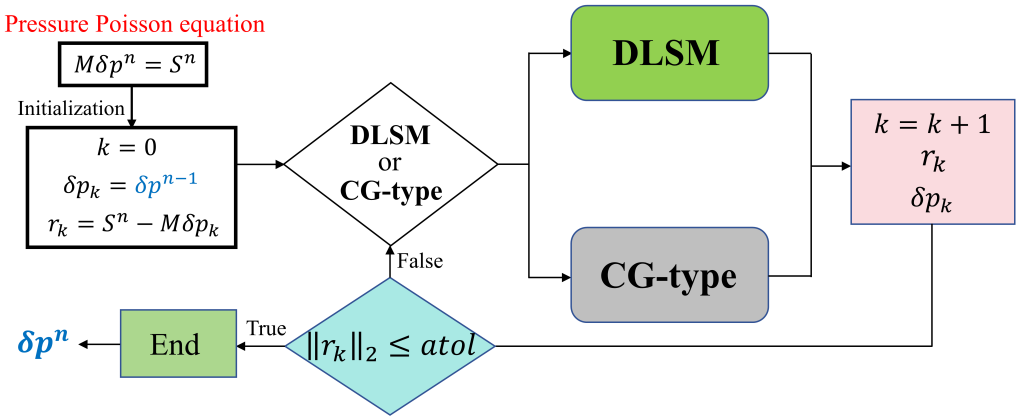


Figure 1: The workflow of HyDEA, n denotes the time step for an unsteady CFD simulation, k represents the iteration index for solving the discrete PPE at the current time step, and $atol$ is the predefined absolute tolerance. The initial value for iteration is taken from last time step: $\delta p_k = \delta p^{n-1}$

Further details of HyDEA are illustrated in Fig. 2. As which solver executed first is potentially essential, HyDEA offers two distinct choices as indicated by the red-dashed circle in Fig. 2(a). The two implementations in Fig. 2(b) and Fig. 2(c) correspond to the two versions of HyDEA for *one round* of the hybrid algorithm: the former represents a CG-type method as the initial solver, while the latter employs DLSM as the initial solver. Moreover, k represents the current iteration number, and *mod* is the modulo operation. For the two respective methods $DLSM_{count}$ and $CG-type_{count}$ record the numbers of consecutive iterations while Num_{DLSM} and $Num_{CG-type}$ denote the predefined maximum consecutive iterations.

2.4. Dataset and model configuration

2.4.1. Dataset construction

Following the method proposed by Kaneda et al. [32], random vectors biased toward the lower end of the spectrum of M are constructed as the training dataset. This dataset construction method is fundamentally rooted in the convergence behavior of the CG method, for which we provide a comprehensive analysis in Appendix A.

The critical step in dataset construction is to generate a set of approximate eigenvectors $Q_m = [\mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_{m-1}] \in R^{a\times m}$ (where $m < a$), which can effectively represent the entire spectrum of M. Specifically, the Lanczos iteration first transforms M to a low-dimensional tridiagonal matrix $T_m \in R^{m\times m}$ [50], yielding the relation

$$T_m = V_m^T M V_m, \tag{39}$$

where $V_m \in R^{a\times m}$ is the orthogonal matrix whose columns consist of the orthogonal vectors generated during the Lanczos iteration. Subsequently, the eigenvalues of T_m form a diagonal matrix $\Lambda_m \in R^{m\times m}$ with non-decreasing eigenvalues on the diagonal referred to as approximate eigenvalues or Ritz values of M. Moreover, the eigenvectors of T_m form a matrix $U_m \in R^{m\times m}$. Thereafter, $Q_m = V_m U_m$ forms the approximate eigenvectors of M, commonly known as Ritz vectors. Finally,

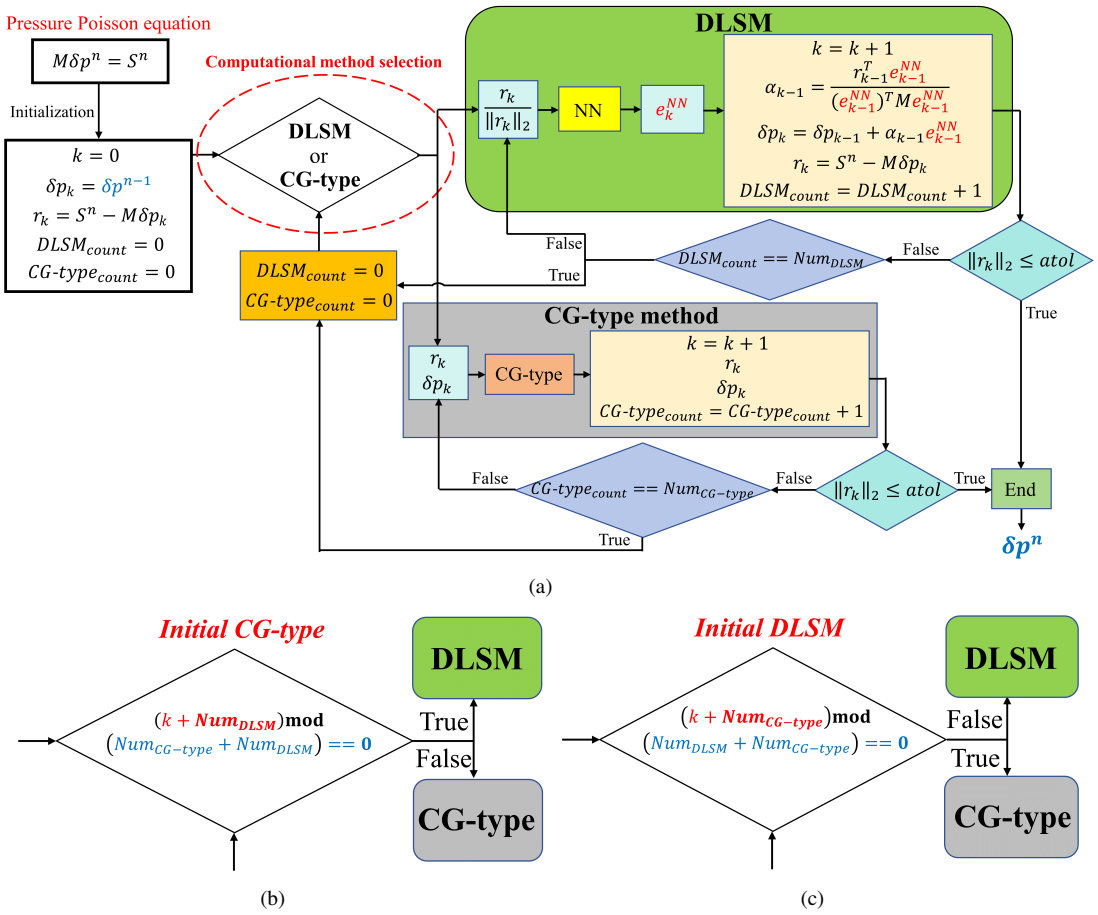


Figure 2: The implementation details and two versions of HyDEA . (a) Detailed workflow. (b) For each round of the hybrid algorithm, a CG-type method is taken as the initial solver for maximum $Num_{CG-type}$ iterations followed by DLSM for maximum Num_{DLSM} iterations. (c) For each round of the hybrid algorithm, DLSM is taken as the initial solver for maximum Num_{DLSM} iterations followed by a CG-type method for maximum $Num_{CG-type}$ iterations.

the dataset is constructed as follows

$$r^i = \frac{\sum_{j=0}^{m-1} c_j^i \mathbf{q}_j}{\| \sum_{j=0}^{m-1} c_j^i \mathbf{q}_j \|_2}, \tag{40}$$

$$c_j^i = \begin{cases} 9 \cdot N(0,1), & \text{if } 0 \le j \le b \cdot m \\ N(0,1), & otherwise \end{cases}, \tag{41}$$

where i denotes distinct indices for diverse data, $N(0,1)$ is a random number drawn from the normal distribution. The hyper-parameter m denotes the number of Lanczos iterations, which directly determines the number of approximate eigenvalues and eigenvectors computed for M. The hyper-parameter b controls the frequency characteristics of vectors r in the dataset. Specifically, decreasing the value of b enhances the relative dominance of low-frequency components in the dataset. Since values of m and b may affect the quality of the dataset and the overall performance of HyDEA, we shall perform a sensitivity analysis of the two parameters.

2.4.2. *Deep operator network and model training*

The mapping between e_k and r_k in Eq. (38) can be understood as an operator, which we employ the DeepONet to learn its biased version governed by the dataset. The unstacked version of DeepONet employed here consists of a branch network and a trunk network [16] as illustrated in Fig. 3, where the branch network is a U-Net architecture made of convolutional neural networks (CNN) [51] concatenated by a feedforward neural network (FNN) and the trunk network is a simple FNN. $[bs, c, h, w]$ is the feature map, where bs is *batchsize*, c represents the number of channels, h and w are the

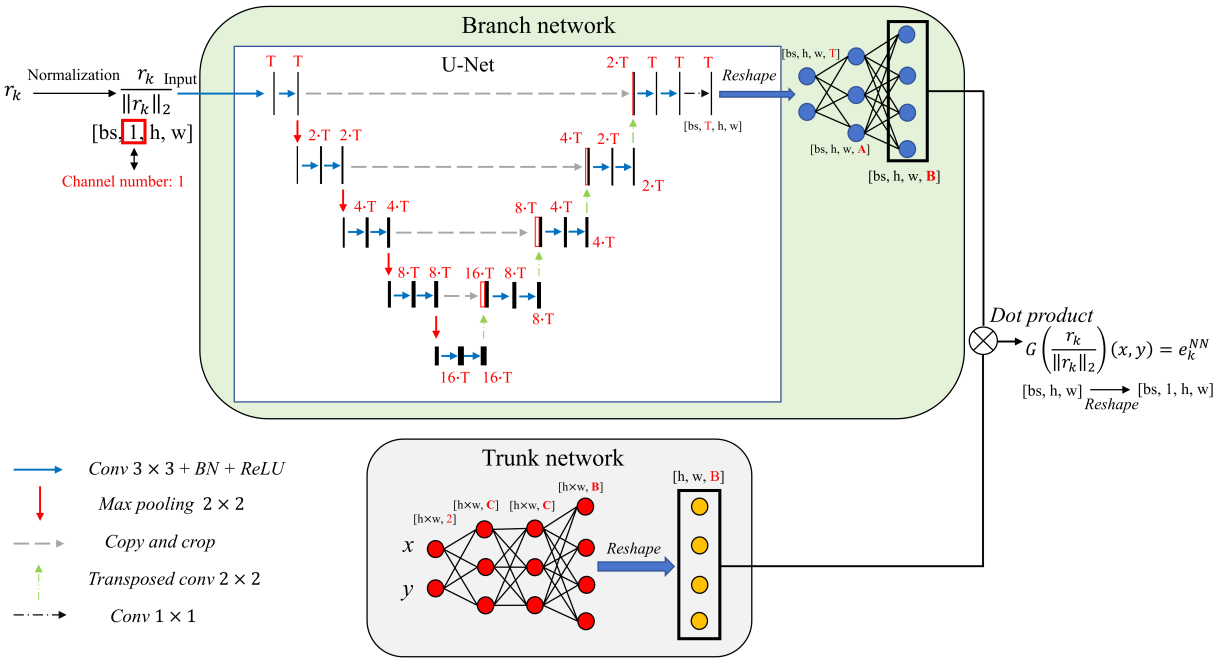


Figure 3: The DeepONet architecture.

sizes of the input in two spatial directions. In U-Net, T, $2 \cdot T$, $4 \cdot T$, etc., represent the numbers of feature map channels at specific network layers and $T = 40$ by default. A, B and C are the number of neurons in FNNs and are assigned as 100, 200 and 100 by default, respectively. A detailed analysis of these parameters are presented in Appendix B.

Any vector r constructed by Eq. (40) serves as input to the DeepONet, which predicts a vector $NN(r)$ governed by the loss function as

$$Loss = \frac{1}{N} \sum_{i=1}^{N} [r_i - (M \cdot NN(r))_i]^2, \tag{42}$$

where N is the total number of the grid points. This loss function is more concise than the form given by Kaneda et al. [32] and is similar to the form suggested in Lan et al. [52]. The random mini-batch [53] method is employed, with $bs = 20$. The training process consists of 1000 epochs and each epoch contains 1000 iterations. The size of the training dataset is always $54,000$ irrespective of CFD resolutions. The Adam optimizer combined with sharpness aware minimization method (SAM) [54] are utilized to update the neural network parameters. The cosine learning rate schedule is utilized and set to 0.0002. The maximum parameter perturbation range in SAM is set to be 0.0002 [21].

Table 1: Components and associated resolutions of HyDEA (CG-type+DLSM) and (CG-type+DCDM)

HyDEA (CG-type+Deep learning)	DLSM-1	DLSM-2	DCDM-1	DCDM-2
resolution	128×128	192×192	128×128	192×192
CG	✓	✓	-	-
ICPCG	✓	✓	-	✓
JPCG	✓	✓	-	-
MGPCG	✓	✓	-	-
super-resolution	$640 \times 640 \downarrow$	-	$512 \times 512 \downarrow$	-
solid structures	-	✓	-	✓

2.5. Technical details

We summarize components, resolutions and applications of HyDEA in Table 1. Each realization of HyDEA consists of a CG-type method and a DLSM or a DCDM. CG-type methods include the vanilla CG and three preconditioned variants, namely, incomplete Cholesky decomposition PCG (ICPCG), Jacobi PCG (JPCG), and Multigrid PCG (MGPCG). Moreover, both DLSM and DCDM have two resolutions:

- DLSM-1 (or DCDM-1) with DeepONet receiving input feature maps of a spatial resolution of 128×128;
- DLSM-2 (or DCDM-2) with DeepONet receiving input feature maps of a spatial resolution of 192×192.

For example, HyDEA (CG+DLSM-1) means that HyDEA hybridizes the vanilla CG method with DLSM for the first resolution. The same acronym rule applies to HyDEA (ICPCG+DLSM-1), HyDEA (JPCG+DLSM-1), and HyDEA (MGPCG+DLSM-1). It also applies to four variants of HyDEA (xxCG+DLSM-2) for the second resolution.

The same nomenclature also applies to the hybrid algorithm of a CG-type with DCDM, namely, HyDEA (ICPCG+DCDM-2) for the second resolution.

DLSM, DCDM and associated DeepONet are implemented using `Python` and `PyTorch`. CG-type methods are adopted from the `PETSc` library in *C* [55] with its `Python` interface `petsc4py`.

The interface between `PetIBM` and `Python` is implemented using the `Pybind11` library [56].

The DeepONet models are trained and deployed on a single NVIDIA GeForce RTX 4090 and all other computations are executed on a single CPU of Intel Xeon Silver 4210R.

3. Results and discussions

We shall systematically evaluate the performance of HyDEA through four cases of benchmarks governed by the incompressible fluid dynamics.

Case 1: Two-dimensional (2D) lid-driven cavity flows are considered in Section 3.1 and more specifically

- in Section 3.1.1 HyDEA (CG-type+DLSM-1) is evaluated for $Re = 1000$.
- in Section 3.1.2 HyDEA (CG-type+DLSM-2) is evaluated for $Re = 3200$.
- in Section 3.1.3 DLSM-1 further applies in mismatched-discretization scenarios via bilinear interpolation upto resolution 640×640, and HyDEA (CG-type+DLSM-1) is evaluated for $Re = 10,000$ as a super-resolution application.

In Cases 2-4, the decoupled immersed boundary projection method (DIBPM) is activated and HyDEA is evaluated for fluid-structure interactions with one stationary circular cylinder in Section 3.2, two stationary elliptical cylinders in Section 3.3, and one moving cylinder in Section 3.4, respectively. Without re-training or fine-tuning of network parameters, DLSM-2 applies universally in all three cases and the generalization capacity of HyDEA (CG-type + DLSM-2) is evaluated.

As presented in Section 2.3, there are two sequential orders for one round of HyDEA algorithm and they are

1. A CG-type method is taken as the initial solver and followed by DLSM shown Fig. 2(b);
2. DLSM is taken as the initial solver and followed by a CG-type method shown in Fig. 2(c).

We will employ the first version for most of the applications due to its better robustness. The reasoning is as follows: the initial residual vector r_0 is arbitrary and does not necessarily tend to the lower end of the spectrum of M, compromising the efficacy of neural network-based computation, as in the second version of HyDEA. In contrast, the first version of HyDEA adopts a CG-type method during initial iterations, which results in a "smoothed" residual to boost the strength of neural network for later iterations. We defer a fair comparison of the two versions in the context of a complex flow scenario with one moving cylinder in Section 3.4.

3.1. Case 1: 2D lid-driven cavity flow

3.1.1. $Re = 1000$

The geometric configuration and boundary conditions of the flow are illustrated in Fig. 4. The computational domain is a square with side length $H_x = H_y = H = 1$. The x-component of the velocity for the top lid is set to be $u = 1$ and all other velocity components of the boundaries are 0. The kinematic viscosity ν is 0.001, and the CFD time step $\Delta t = 0.004$. Therefore, Reynolds number $Re = uH/\nu = 1000$. The computational domain is discretized using a uniform Cartesian grid with resolution of 128×128. Consequently, the size of the input feature map for the branch network of DeepONet is set to $h = 128$ and $w = 128$.

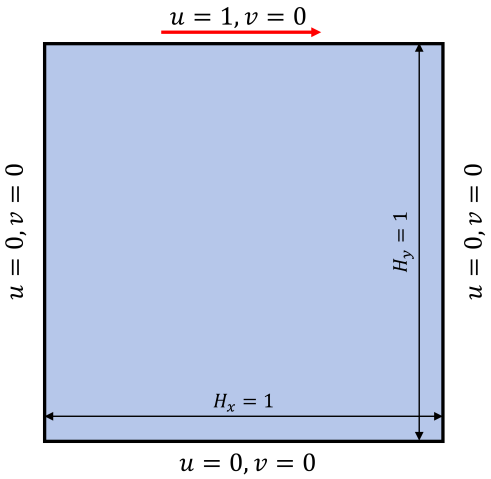


Figure 4: Schematic diagram of 2D lid-driven cavity flow.

To prepare for the training dataset, we adopt the parameter values $m = 3000$ and $b = 0.6$. A systematic sensitivity analysis of their influence on HyDEA's performance is presented in Appendix C. Furthermore, we set $Num_{CG-type} = 3$ and $Num_{DLSM} = 2$ as the maximum executive iterations of the two respective solvers in each round of the hybrid algorithm and remove a detailed analysis on the two values into Appendix D. The termination residual expressed in $L2$-norm for the iterative calculation is set to $\epsilon = 10^{-6}$, being consistent with the official benchmark of `PetIBM`.

The iterative residuals of the PPE using both HyDEA (CG-type method+DLSM-1) and various CG-type methods are compared at three representative time steps in Fig. 5. For example, in Fig. 5(a), (b) and (c) the iterative residuals using CG method alone provide baseline performance in (blue) solid lines, where it takes over 250, 70 and 45 iterations to achieve the predefined tolerance at $t = 10\Delta t$, $100\Delta t$ and $1000\Delta t$, respectively. The required number of iterations decreases as the flow develops towards the steady state. At $t = 10\Delta t$, HyDEA receives a similar initial residual (shown in solid circle) as in CG, then executes $Num_{CG-type} = 3$ CG iterations (solid pentagon), followed by $Num_{DLSM} = 2$ DLSM-1 iterations (empty pentagon); and again $Num_{CG-type} = 3$ CG iterations and then $Num_{DLSM} = 2$ DLSM-1 iterations. In total, HyDEA reaches the predefined tolerance by 2 rounds of the hybrid algorithm with 10 iterations in total, a significant reduction compared to 250 iterations of the CG method alone. Similarly, HyDEA finishes in 1 round of the hybrid algorithm for $t = 100\Delta t$ with 5 iterations in total and in less then 1 round for $t = 1000\Delta t$ with 4 iterations in total. Note that at $t = 1000\Delta t$, the initial residual of HyDEA appears apparently different from that of the CG method, as they have a completely different history of residuals from previous time steps.

Furthermore, we conduct similar comparisons between three standalone preconditioned CG methods: ICPCG, JPCG, and MGPCG with HyDEA integrating the corresponding PCG methods. As shown in Fig. (5)(d)-(l), HyDEA always reaches the predefined tolerance with maximum 2 rounds of the hybrid algorithm or maximum 10 iterations, an overall significant reduction of iteration numbers compared to their counterparts.

Table 2 summarizes the wall-time acceleration ratio and total computational time for solving the PPE by HyDEA relative to the corresponding CG-type methods over $10,000$ consecutive time steps. Overall HyDEA (CG-type+DLSM-1) demonstrates a noticeable acceleration compared to its counterpart CG-type variant alone.

Table 2: The wall-time acceleration ratio and computational time for solving the PPE by HyDEA for a duration of $10,000\Delta t$ at $Re = 1000$.

HyDEA	(CG+DLSM-1)	(ICPCG+DLSM-1)	(JPCG+DLSM-1)	(MGPCG+DLSM-1)
Acceleration ratio	×3.83	×1.88	×3.56	×1.64
computational time (s)	71	83	75	167

Some comments are in order; the reduction of computational time in Table 2 is less impressive than the reduced number of iterations in Fig. 5. This performance discrepancy stems from multiple factors. Foremost, HyDEA requires

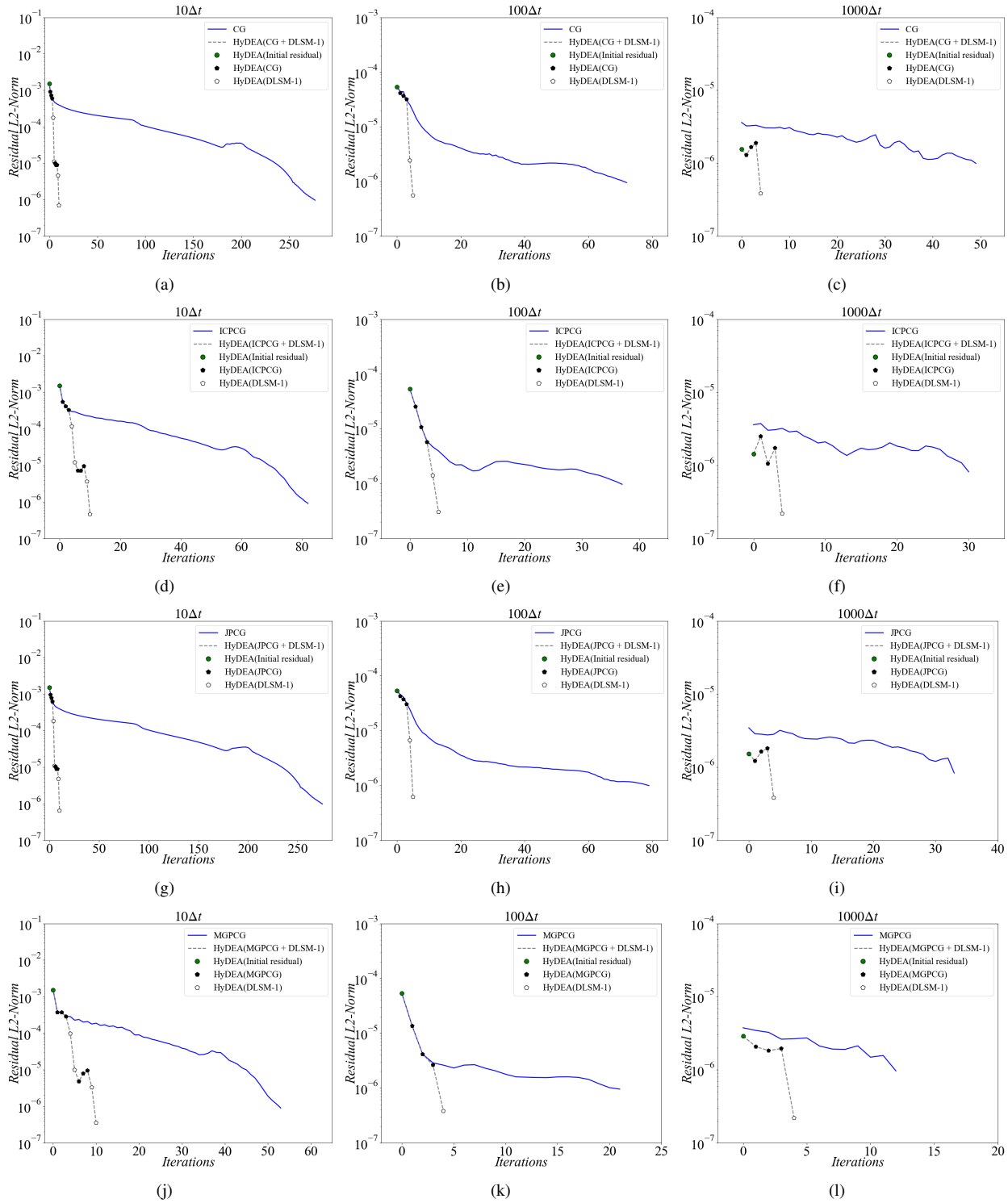


Figure 5: The iterative residuals of solving the PPE at the 10*th*, 100*th* and 1000*th* time steps for a 2D lid-driven cavity flow at $Re = 1000$. (a)-(c) HyDEA (CG+DLSM-1). (d)-(f) HyDEA (ICPCG+DLSM-1). (g)-(i) HyDEA (JPCG+DLSM-1). (j)-(l) HyDEA (MGPCG+DLSM-1).

frequently bidirectional data tensor transfers between GPU (neural networks) and CPU (iterative methods), leading to significant execution delays due to communications. Secondly, the DLSM method is implemented in `Python`, which inherently limits its computational efficiency due to the interpreted nature of the language. Additionally, the current network architecture may be optimized for faster inference. While these technical factors limit the current computational efficiency of HyDEA, they also represent a clear opportunity for future optimization. We emphasize that the present work primarily focuses on establishing the algorithmic framework of HyDEA and examine its generalization capability.

Taking the ICPCG and HyDEA (ICPCG+DLSM-1) as an example, velocity profiles at steady state are compared with the results of Ghia et al. [57], as illustrated in Fig. 6. The excellent agreement demonstrates a high accuracy of present computational results. Furthermore, Fig. 7 presents the velocity fields at the $1500th$ and $10,000th$ time steps, confirming that the temporal evolution of the flow field is accurately captured.

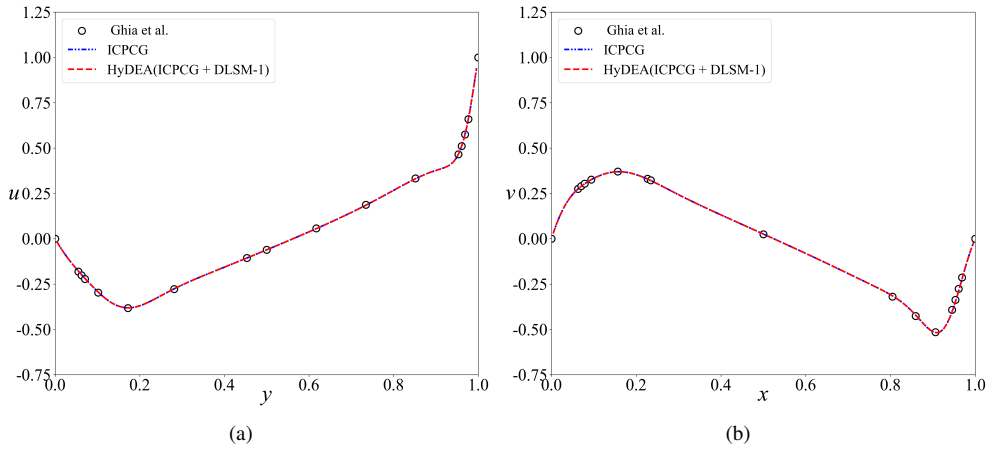


Figure 6: Profiles of u along $x = 0.5$ and v along at $y = 0.5$ at steady state for 2D lid-driven cavity flow at $Re = 1000$ by ICPCG and HyDEA (ICPCG+DLSM-1). (a) u. (d) v.

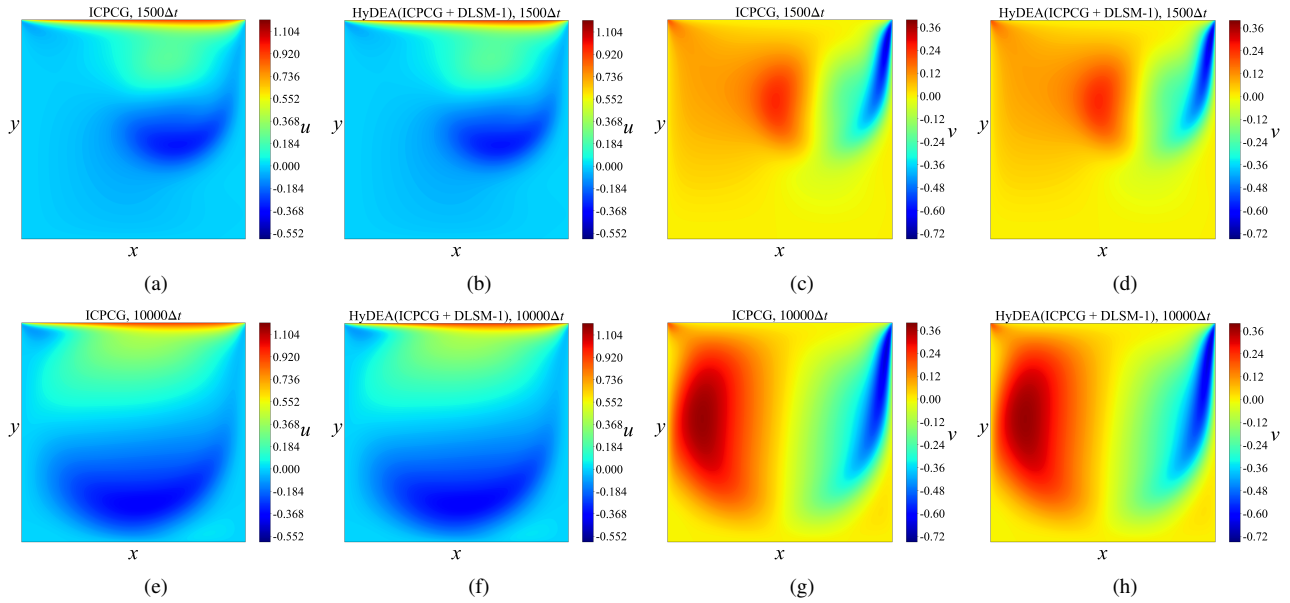


Figure 7: Velocity fields for 2D lid-driven cavity flow at $Re = 1000$ by ICPCG and HyDEA (ICPCG+DLSM-1). (a)-(d) u and v at the $1500th$ time step. (e)-(h) u and v at the $10,000th$ time step.

3.1.2. Re = 3200

Based on Section 3.1.1, we adjust the configuration with $\nu = 0.0003125$ and $\Delta t = 0.002$ so that $Re = 3200$. To accommodate the change, the grid resolution becomes 192×192 and consequently, the size of the input feature map for the branch network of DeepONet is set to $h = 192$ and $w = 192$.

To prepare for the training dataset, we adopt the parameter values $m = 7000$ and $b = 0.6$. A systematic sensitivity analysis of their influence on HyDEA's performance is also provided in Appendix C. Furthermore, we set $Num_{CG-type} = 3$ and $Num_{DLSM} = 2$ as the maximum executive iterations of the two respective solvers in each round of the hybrid algorithm. The termination criterion is again with $\epsilon = 10^{-6}$.

To avoid over-extensive comparisons, we only present results of HyDEA (CG+DLSM-2), HyDEA (ICPCG+DLSM-2) and their counterparts. The iterative residuals of the PPE solution at the $10th$, $100th$ and $1000th$ time steps are shown in Fig. 8. Overall HyDEA reaches the predefined toleration within 4 rounds of the hybrid algorithm with maximum around 16 iterations in total. HyDEA demonstrates significantly reduced number of iterations compared to CG-type methods alone, being consistent with the results in Section 3.1.1.

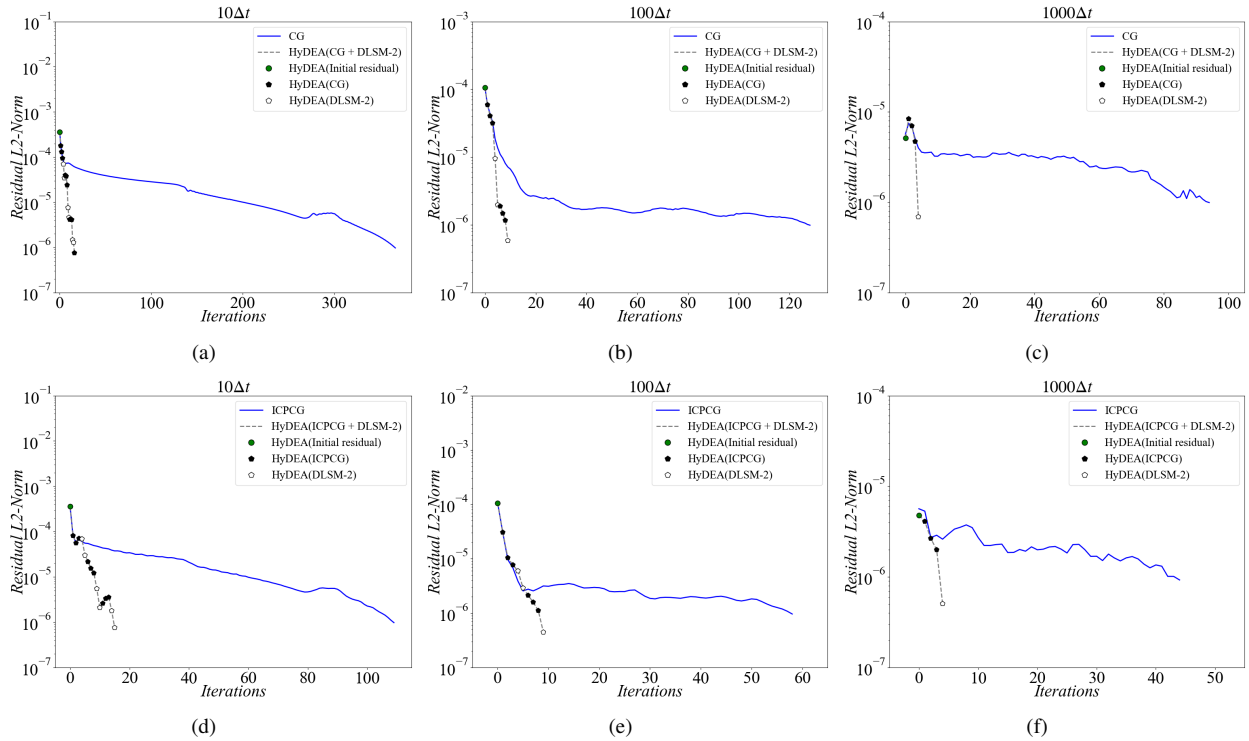


Figure 8: The iterative residuals of solving the PPE at the $10th$, $100th$ and $1000th$ time steps for 2D lid-driven cavity flow at $Re = 3200$. (a)-(c) HyDEA (CG+DLSM-2). (d)-(f) HyDEA (ICPCG+DLSM-2).

Table 3 summarizes the wall-time acceleration ratio and computational time of solving the PPE by HyDEA relative to the corresponding CG-type methods over $10,000$ consecutive time steps. Overall HyDEA (CG-type+DLSM-2) demonstrates a noticeable acceleration compared to its counterpart CG-type variant alone. Notably, the acceleration ratios become more significant than those for the low-resolution in Section 3.1.1, highlighting HyDEA's enhanced potential for high-resolution flow simulations.

Table 3: The wall-time acceleration ratio and computational time of solving the PPE by HyDEA for a duration of $10,000\Delta t$ at $Re = 3200$.

HyDEA	(CG+DLSM-2)	(ICPCG+DLSM-2)	(JPCG+DLSM-2)	(MGPCG+DLSM-2)
Acceleration ratio	×5.96	×2.82	×5.64	×2.08
computational time (s)	131	162	134	342

Furthermore, as a representative case, HyDEA (ICPCG+DLSM-2) demonstrates excellent agreement with the benchmark results of Ghia et al. [57], as evidenced by the steady-state velocity profiles in Fig. 9. The velocity contours at the $1500th$ and $10,000th$ time steps in Fig. 10 further validate HyDEA's accuracy, capturing the flow fields with high fidelity.

3.1.3. Performance in mismatched discretizations

In light of the significant computational overhead associated with dataset construction and neural network training for high-resolution simulations, this section examines HyDEA's applicability for super-resolutions. Taking the version of Fig. 2(b)) for example, where a CG-type method is taken as the initial solver, the workflow of HyDEA for super-resolution is sketched as follows:

1. A DeepONet is trained *offline* based on a low-resolution dataset;
2. A CG-type method conducts $Num_{CG-type}$ iterations on a high-resolution;
3. The DLSM module receives the residual r_k from the CG-type method and performs a downsampling to low-resolution r_k^{low};
4. The DeepONet predicts e_k^{NN-low} for r_k^{low} and performs an upsampling to high-resolution e_k^{NN};

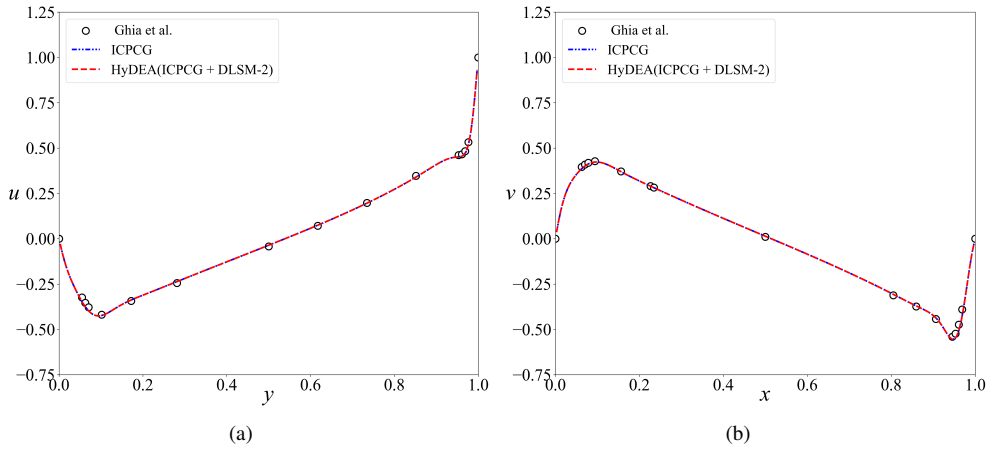


Figure 9: Profiles of u along $x = 0.5$ and v along $y = 0.5$ at stedy state for 2D lid-driven cavity flow at $Re = 3200$ by ICPCG and HyDEA (ICPCG+DLSM-2). (a) u. (b) v.

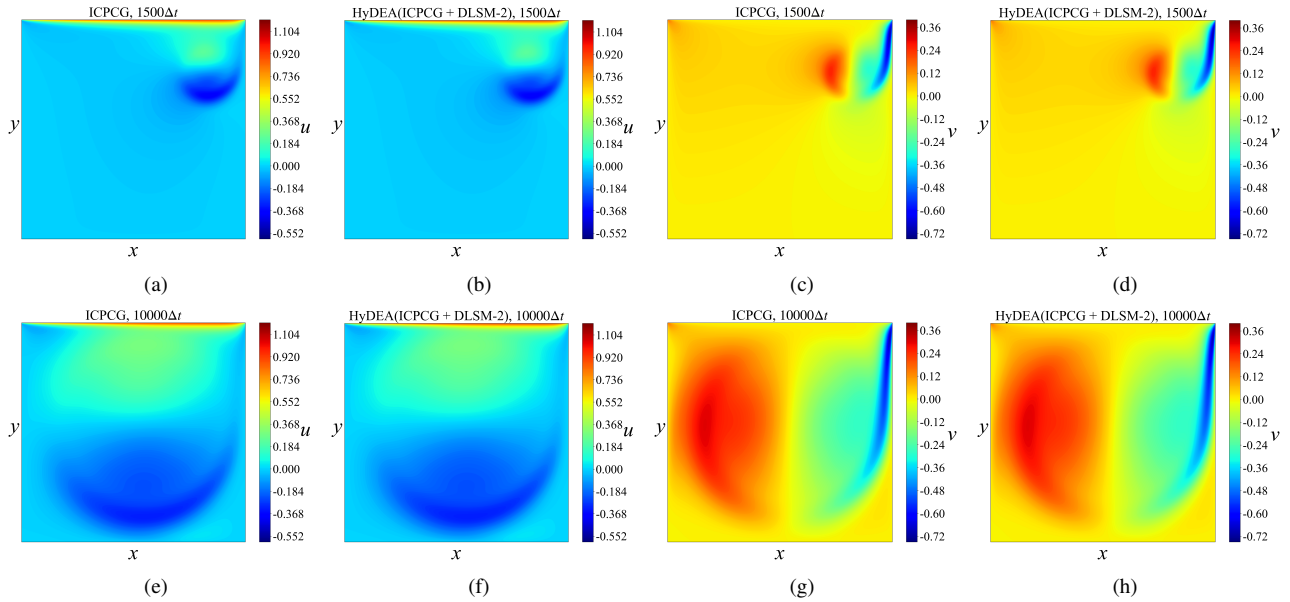


Figure 10: Velocity fields for 2D lid-driven cavity flow at $Re = 3200$ by ICPCG and HyDEA (ICPCG+DLSM-2). (a)-(d) u and v at the $1500th$ time step. (e)-(h) u and v at the $10,000th$ time step.

5. The DLSM utizlied the e_k^{NN} to finish the iteration.

This HyDEA procedure repeats until convergence during each time step of the CFD. We name this modified framework as super-resolution (SR)-HyDEA for later reference. To enable this capability, a pre-processing step and a post-processing step are introduced for DeepONet within the DLSM, as illustrated in Fig. 11. The input feature map is first downsampled via bilinear interpolation from high-resolution ($[1,1,p,q]$) to low-resolution ($[1,1,h,w]$) format before being fed into the DeepONet. Subsequently, the DeepONet's output is upsampled back to the original high resolution ($[1,1,p,q]$) via bilinear interpolation, after which the computation proceeds.

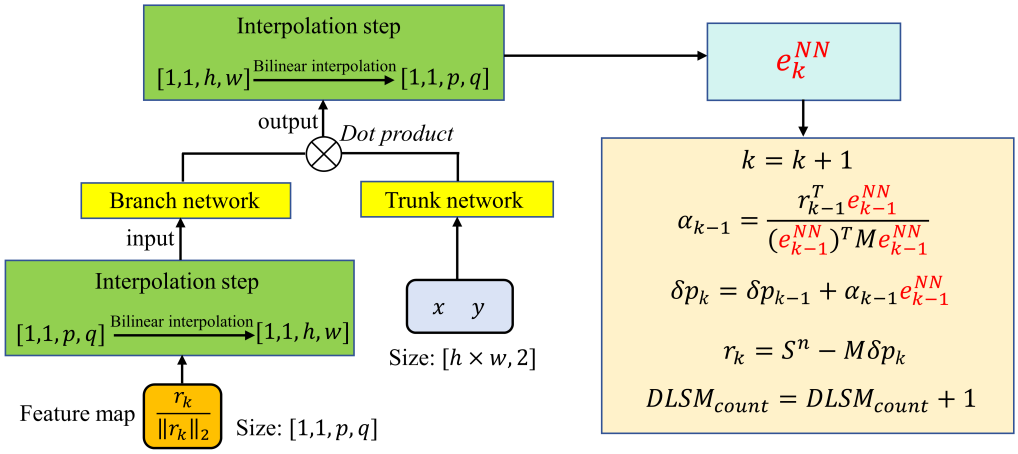


Figure 11: Downsmapling and upsampling for DeepONet within DLSM.

As demonstration, we utilize DLSM-1 in Section 3.1.1 for SR-HyDEA, where the DeepONet receives input feature maps with low resolution $[1,1,128,128]$, corresponding to a CFD grid resolution of 128×128. $Num_{CG-type} = 3$, and Num_{DLSM} varies between 1 and 2. We evaluate the performace of this SR-HyDEA for four higher resolutions: 256×256, 384×384, 512×512 and 640×640. The CFD time step Δt adjusts accordingly to be 0.002, 0.002, 0.0015 and 0.0012. The geometric configuration and boundary conditions remain consistent with Section 3.1.1, but we set $\nu = 0.0001$ to achieve $Re = 10,000$.

Here, ICPCG is taken as the baseline performance and compared against SR-HyDEA (ICPCG+DLSM-1): the iterative residuals for solving the PPE at the $10th$, $100th$ and $1000th$ time steps with $Num_{CG-type} = 3$ and $Num_{DLSM} = 1$ are illustrated in Fig. 12. The SR-HyDEA demonstrates a significant reduction in the number of iterations compared to the ICPCG method alone, even when the resolution is increased by $4\times4 = 16$ times in two dimensions. However, the superior performance diminishes when the resolution is further increased to $5 \times 5 = 25$ times, revealing the inherent limitations of low-resolution neural network models in generalizing to high-resolution flow simulations.

Fig. 13 further summarizes the comparison of computational time to solve the PPE over $10,000$ consecutive time steps. In general, SR-HyDEA demonstrates gains in computational efficiency when $Num_{DLSM} = 1$ across the first three high resolutions. However, when $Num_{DLSM} = 2$, its performance benefit almost vanishes at resolution of 512×512. Consequently, we evaluate only for $Num_{DLSM} = 1$ at the resolution of 640×640. The results of computational time are consistent with those of iteration numbers shown in Fig. 12: super-resolution works well if the upscaling factor is ≤ 16, but a more aggressive operation deteriorates the performance.

Fig. 14 presents velocity contours at the $5000th$ and $10,000th$ time steps for the resolution of 512×512. The result demonstrates that SR-HyDEA, despite employing a low-resolution (128×128) DeepONet, achieves solution accuracy comparable to the high-resolution (512×512) ICPCG method.

Comparison of DCDM, DLSM and HyDEA

HyDEA combines neural networks with iterative methods to solve the PPE, while DCDM [32] and DLSM rely exclusively on neural networks. As neural networks are incompetent to accurately capture high-frequency features, the latter two methods may exhibit suboptimal performance in complex flow scenarios. Therefore, an investigation is conducted to evaluate whether the DCDM and DLSM implemented using DeepONet with low-resolution feature maps can effectively simulate high-resolution flow fields through bilinear interpolation, with the corresponding methods designated as SR-DCDM and SR-DLSM, respectively. Fig. 15 illustrates the iterative residuals of solving the PPE at the $1st$ time step for three high resolutions. The iterative residuals of SR-DCDM and SR-DLSM show a modest initial decrease but quickly become plateau during further iterations. This behavior suggests that completely replacing the traditional iterative solver with neural networks is not a suitable choice. In contrast, SR-HyDEA demonstrates a significantly superior performance compared to the SR-DCDM and SR-DLSM.

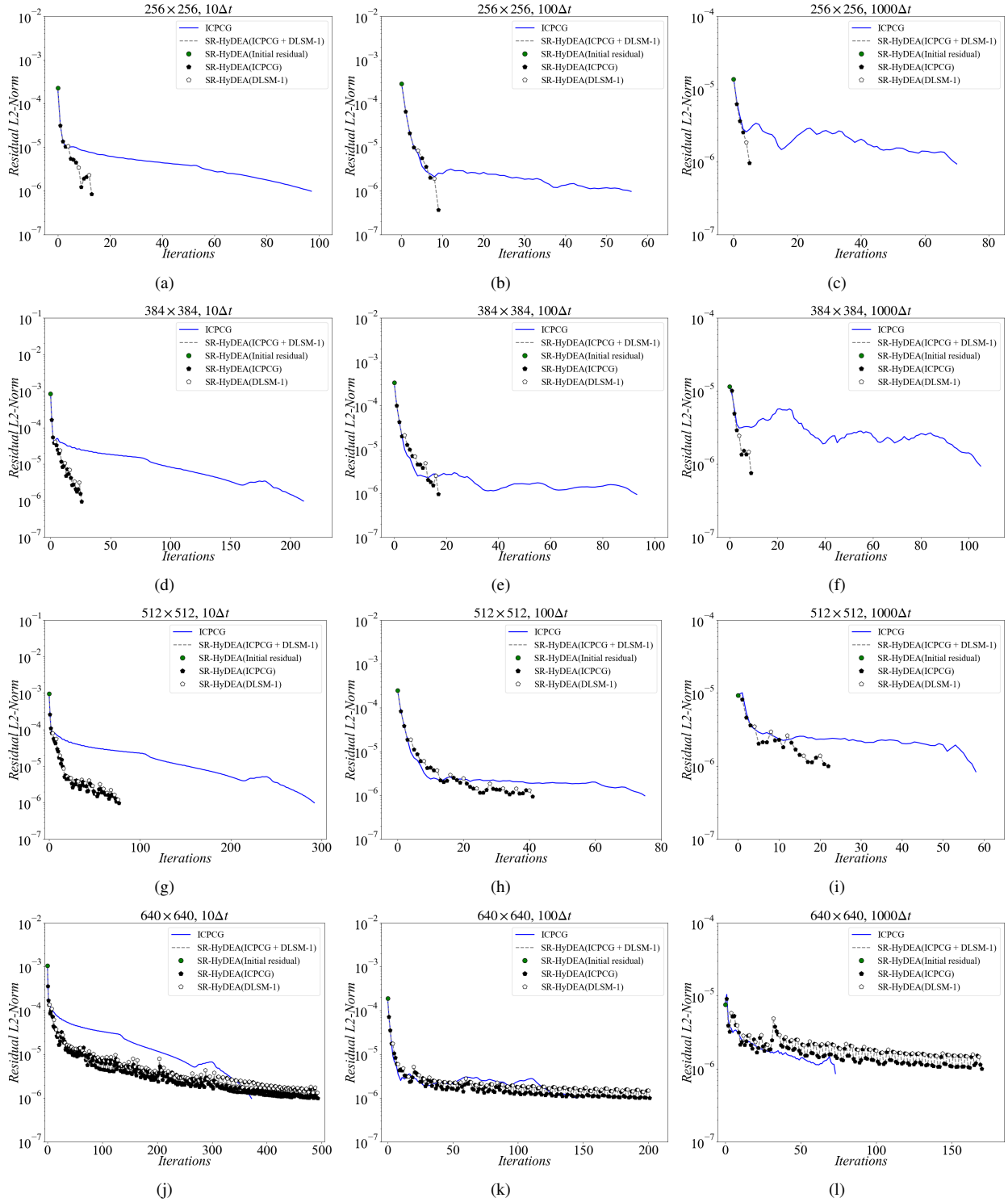


Figure 12: Iterative residuals of solving the PPE for super-resolutions at the 10*th*, 100*th* and 1000*th* time steps for 2D lid-driven cavity flow at $Re = 10,000$. DeepONet is trained *offline* for 128×128 and utilized *online* for (a)-(c) 256×256 (d)-(f) 384×384 (g)-(i) 512×512 (j)-(l) 640×640.

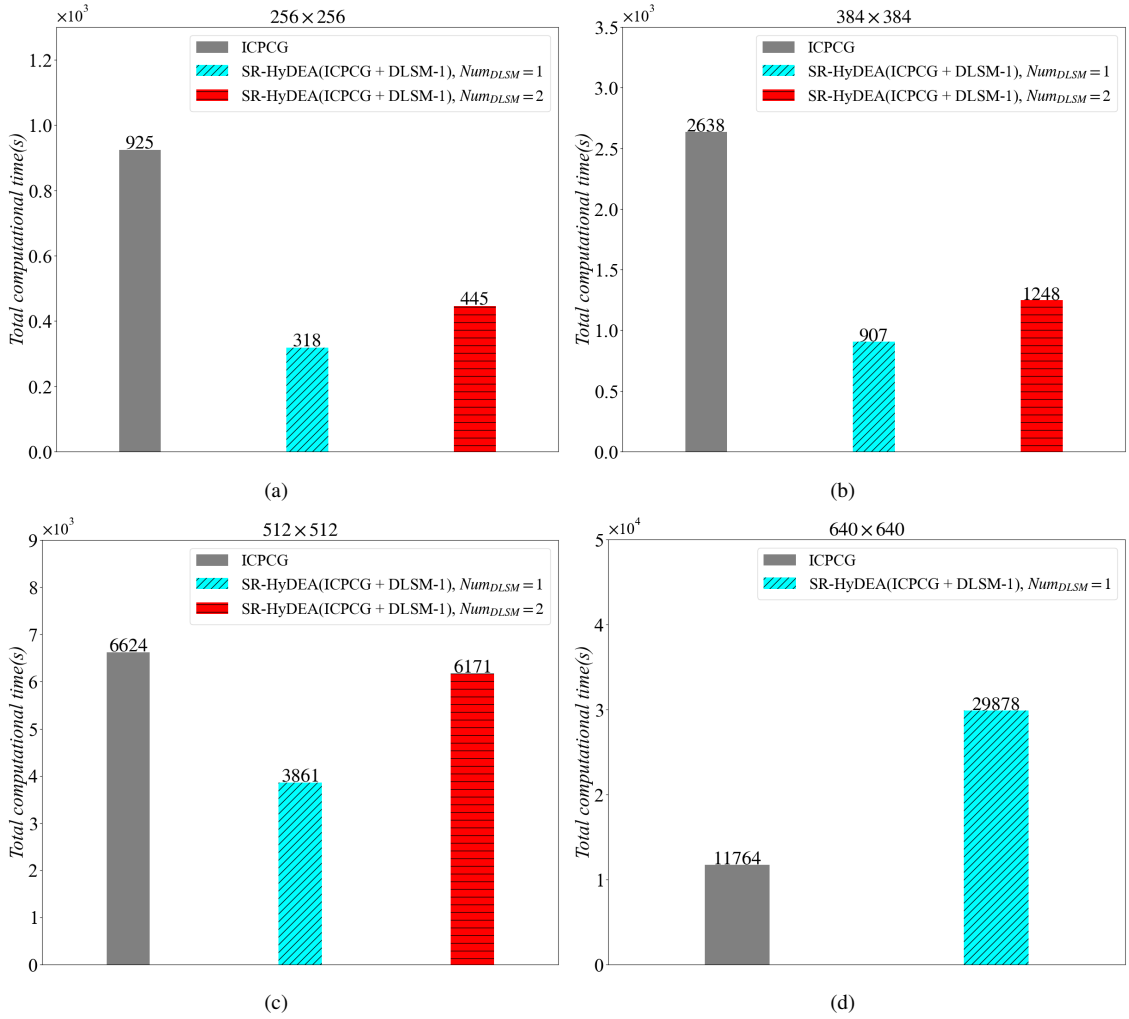


Figure 13: Comparison of computational time required to solve the PPE over a duration of $10,000\Delta t$ for super-resolutions at $Re = 10,000$. (a) 256×256 (b) 384×384 (c) 512×512 (c) 640×640.

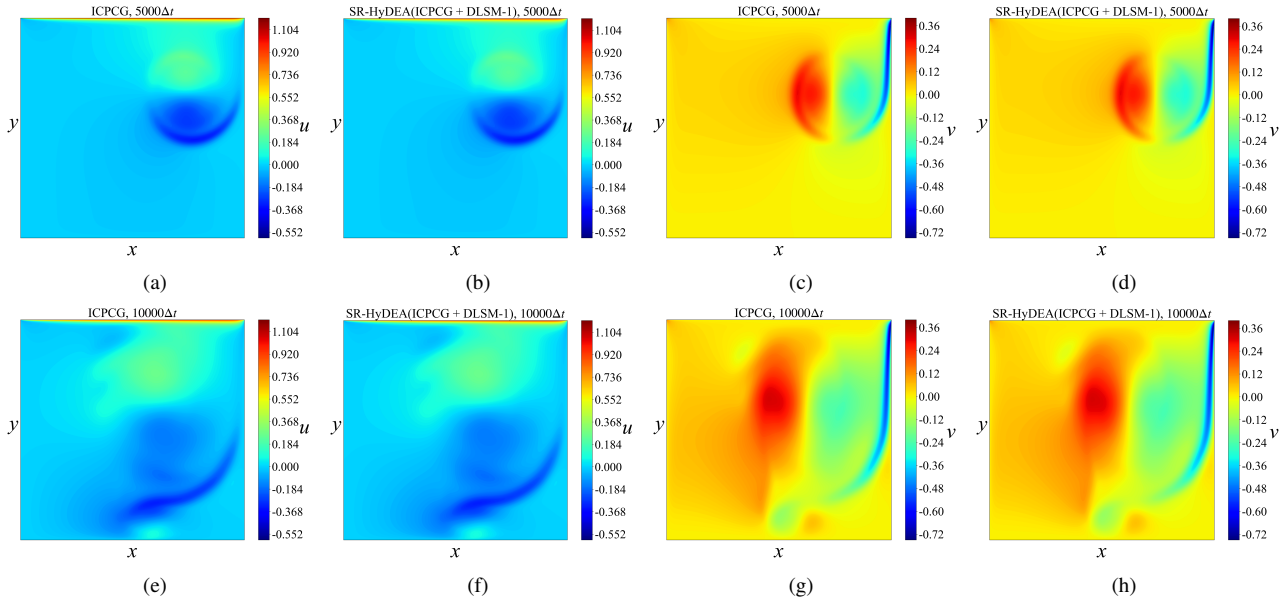


Figure 14: Velocity fields for 2D lid-driven cavity flow at $Re = 10,000$ by ICPCG and SR-HyDEA (ICPCG+DLSM-1) with super-resolution of 512×512. (a)-(d) u and v at the $5000th$ time step. (e)-(h) u and v at the $10,000th$ time step.

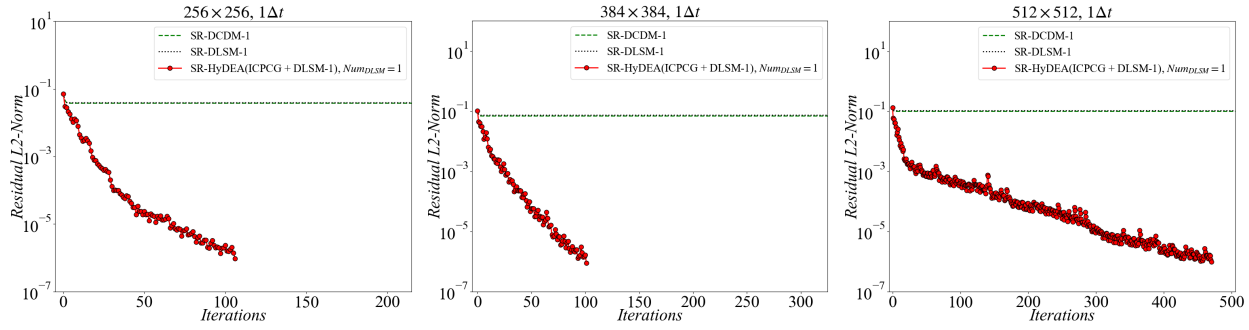


Figure 15: Iterative residuals of solving the PPE for super-resolutions: comparisons among the SR-DCDM-1, SR-DLSM-1 and SR-HyDEA at the first time step for 2D lid-driven cavity flow at $Re = 10,000$. DeepONet is trained *offline* for 128×128 and utilized *online* for (a) 256×256. (b) 384×384. (c) 512×512.

3.2. Case 2: One stationary circular cylinder immersed in 2D lid-driven cavity flow

This section examines the generalizability of HyDEA in simulating flows involving one stationary obstacle. A numerical experiment employs the same setup as in Section 3.1.2 with $Re = 3200$ and grid resolution 192×192, except that a stationary cylinder is immersed in the center of the computational domain as illustrated in Fig. 16.

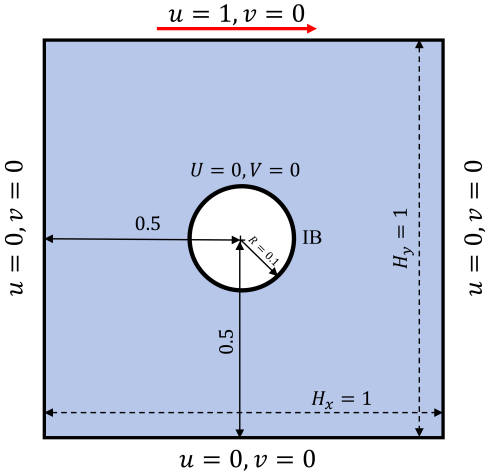


Figure 16: Schematic diagram of 2D lid-driven cavity flow containing an immersed stationary circular cylinder.

Furthermore, the DIBPM is activated for the cylinder in CFD computations and the exactly same DLSM-2 as in Section 3.1.2 is executed in HyDEA with $Num_{CG-type} = 3$ and $Num_{DLSM} = 2$, respectively. We note that *the DeepONet within DLSM-2 remains the same network weights as in Section 3.1.2 and is not retrained.*

Taking HyDEA (ICPCG+DLSM-2) and HyDEA (CG+DLSM-2) as examples, the iterative residuals of solving the PPE at the $10th$, $100th$ and $1000th$ time steps are depicted in Fig. 17. For HyDEA (ICPCG+DLSM-2), it takes 3 rounds of the hybrid algorithm with 15 iterations at $10\Delta t$, 2 rounds with 9 iterations at $100\Delta t$, and 1 round with 4 iterations at $1000\Delta t$, respectively. For HyDEA (CG+DLSM-2), it takes 4 rounds of the hybrid algorithm with 19 iterations at $10\Delta t$, 3 rounds with 14 iterations at $100\Delta t$, and 1 round with 4 iterations at $1000\Delta t$, respectively. In general, HyDEA requires significantly fewer iterations than the ICPCG method alone, being consistent with the single-component flow simulation in Section 3.1.2.

Fig. 18 presents the computational time required to solve the PPE over $10,000$ consecutive time steps using both HyDEA and ICPCG methods. The results clearly indicates that HyDEA needs much less computational time compared to ICPCG method to achieve the same accuracy.

Furthermore, the velocity contours of ICPCG and HyDEA (ICPCG+DLSM-2) at the $1500th$ and $6000th$ time steps are depicted in Fig. 19, demonstrating that the temporal evolution of the flow field is accurately captured.

3.3. Case 3: Two stationary elliptical cylinders immersed in 2D lid-driven cavity flow

This section examines the generalizability of HyDEA in simulating flows involving two stationary obstacles. A numerical experiment employs the same setup as in Section 3.1.2 with $Re = 3200$ and grid resolution 192×192, except that two stationary elliptical cylinders of aspect ratio $4 : 3$ are immersed in the computational domain as illustrated in Fig. 20.

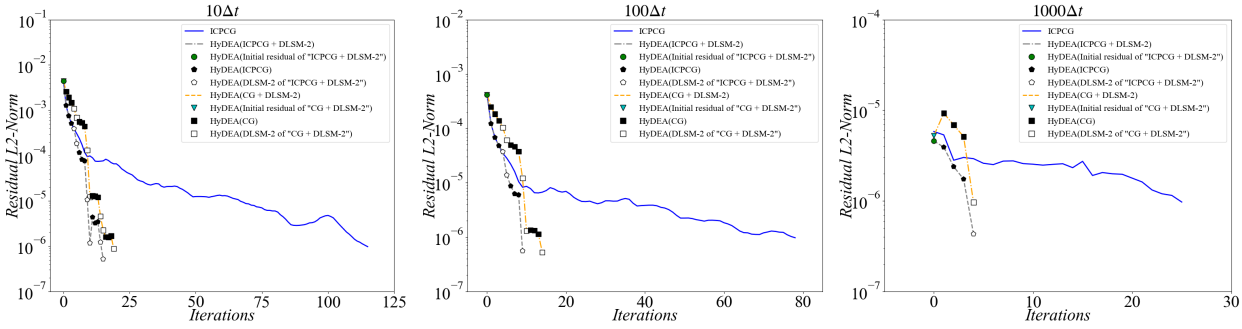


Figure 17: The iterative residuals of solving the PPE for 2D lid-driven cavity flow with an immersed stationary circular cylinder at $Re = 3200$ by ICPCG, HyDEA (ICPCG+DLSM-2) and HyDEA (CG+DLSM-2) . (a) 10*th* time step. (b) 100*th* time step. (c) 1000*th* time step.

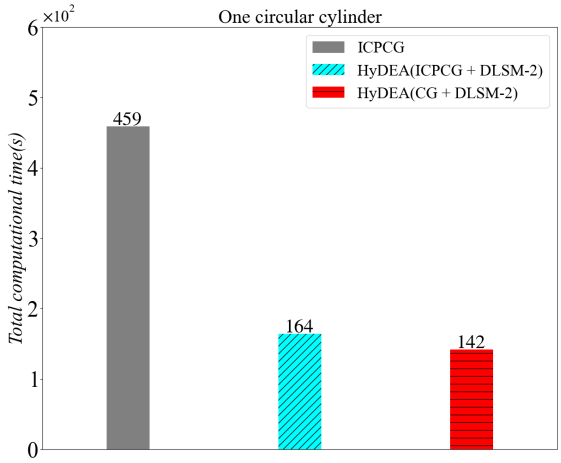


Figure 18: Computational time required to solve the PPE over a duration of $10,000\Delta t$ for 2D lid-driven cavity flow with an immersed stationary circular cylinder at $Re = 3200$.

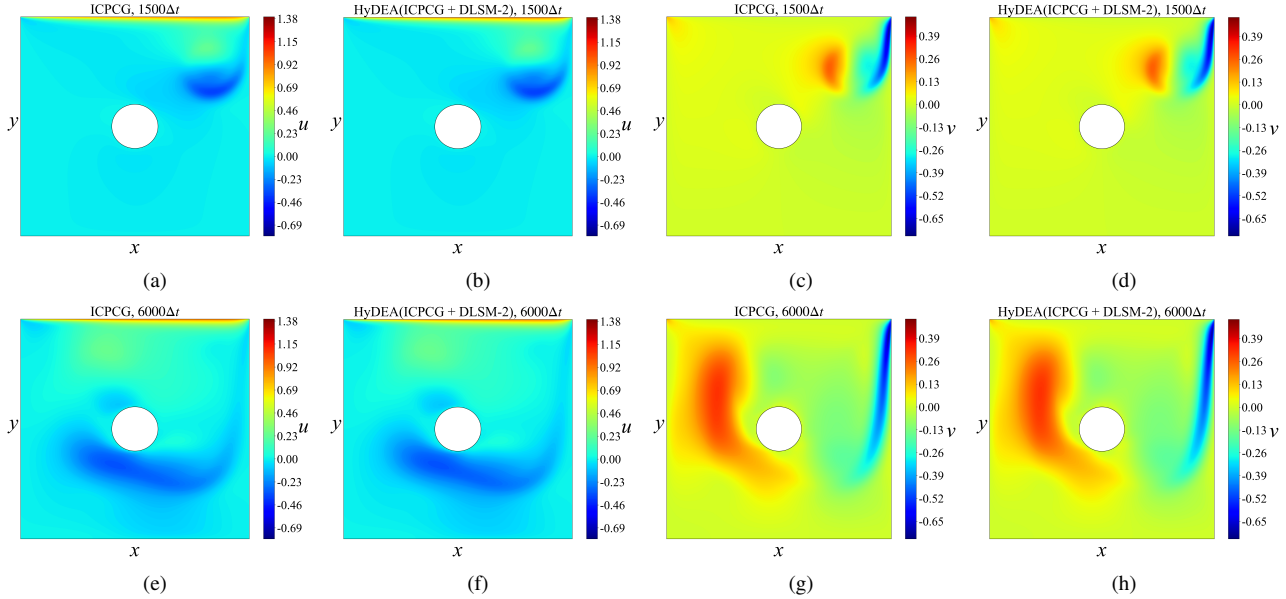


Figure 19: Velocity fields for 2D lid-driven cavity flow with an immersed stationary circular cylinder at $Re = 3200$ by ICPCG and HyDEA (ICPCG+DLSM-2). (a)-(d) u and v at the 1500*th* time step. (e)-(h) u and v at the 6000*th* time step.

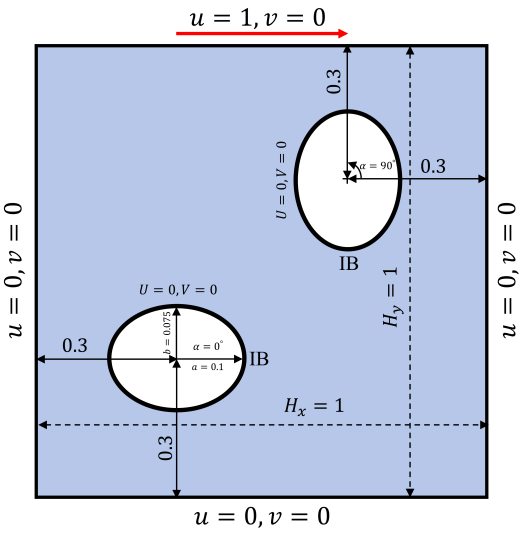
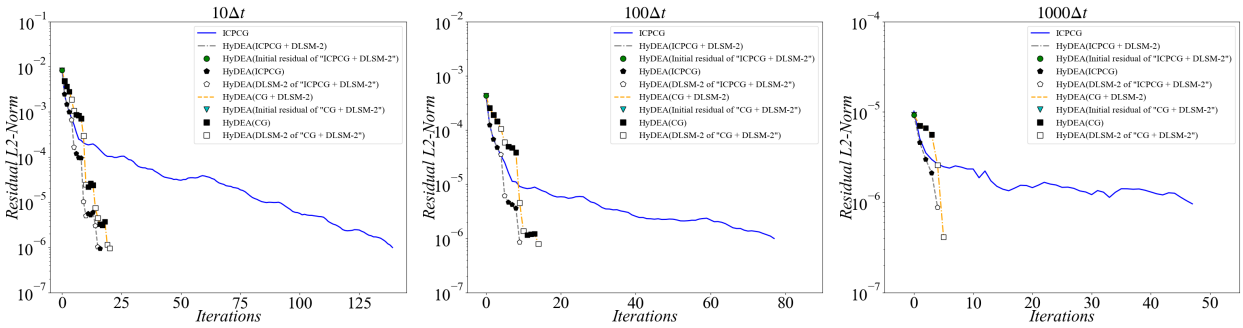


Figure 20: Schematic diagram of 2D lid-driven cavity flow containing two immersed stationary elliptical cylinders. α denotes the angle formed by the major axis of the elliptical cylinder and the x-direction.

Furthermore, the DIBPM is activated for the two elliptical cylinders in CFD computations and the exactly same DLSM-2 as in Section 3.1.2 is executed in HyDEA with $Num_{CG-type} = 3$ and $Num_{DLSM} = 2$, respectively. We note that *the DeepONet within DLSM-2 remains the same network weights as in Sections 3.1.2 and 3.2 and is not retrained.*

Taking HyDEA (ICPCG+DLSM-2) and HyDEA (CG+DLSM-2) as examples, the iterative residuals of solving the PPE at the $10th$, $100th$ and $1000th$ time steps are depicted in Fig. 21. For HyDEA (ICPCG+DLSM-2), it takes 4 rounds of the hybrid algorithm with 16 iterations at $10\Delta t$, 2 rounds with 9 iterations at $100\Delta t$, and 1 round with 4 iterations at $1000\Delta t$, respectively. For HyDEA (CG+DLSM-2), it takes 4 rounds of the hybrid algorithm with 20 iterations at $10\Delta t$, 3 rounds with 14 iterations at $100\Delta t$, and 1 round with 5 iterations at $1000\Delta t$, respectively. In general, HyDEA requires significantly fewer iterations than the ICPCG method alone, being consistent with the single-component flow simulation in Section 3.1.2. The results also align closely with those for one cylinder in Section 3.2, demonstrating HyDEA's robust generalization capability across flows with multiple obstacles.



(a) (b) (c)

Figure 21: The iterative residuals of solving the PPE for 2D lid-driven cavity flow with two immersed stationary elliptical cylinders at $Re = 3200$ by ICPCG, HyDEA (ICPCG+DLSM-2), and HyDEA (CG+DLSM-2). (a) $10th$ time step. (b) $100th$ time step. (c) $1000th$ time step.

Fig. 22 depicts the computational time required to solve the PPE over $10,000$ consecutive time steps using both HyDEA and ICPCG method. The results clearly indicates that HyDEA needs much less computational time compared to ICPCG method to achieve the same accuracy.

Furthermore, velocity contours of ICPCG and HyDEA (ICPCG+DLSM-2) at the $1500th$ and $6000th$ time steps are depicted in Fig. 23, demonstrating that the temporal evolution of the flow field is accurately captured.

3.4. Case 4: 2D flow around an inline oscillating cylinder at Re = 100

This section examines the generalizability of HyDEA in simulating flow around an inline oscillating cylinder. The geometric configuration and boundary conditions are illustrated in Fig. 24 with four walls being stationary. Simulation setup resembles Section 3.1.2 with $\Delta t = 0.002$ and resolution of 192×192. The the trajectory of the cylinder is governed

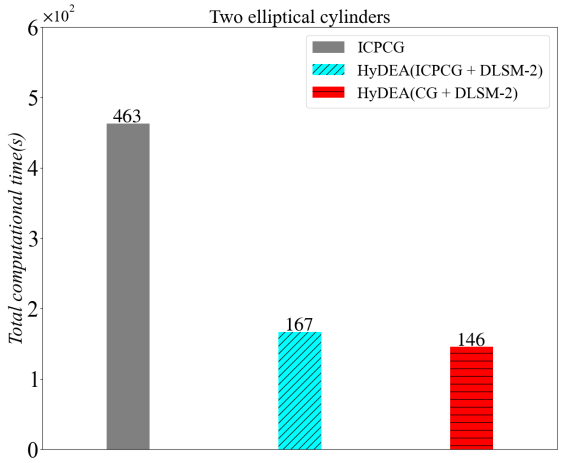


Figure 22: Computational time required to solve the PPE over a duration of $10,000\Delta t$ for 2D lid-driven cavity flow with two immersed stationary elliptical cylinders at $Re = 3200$.

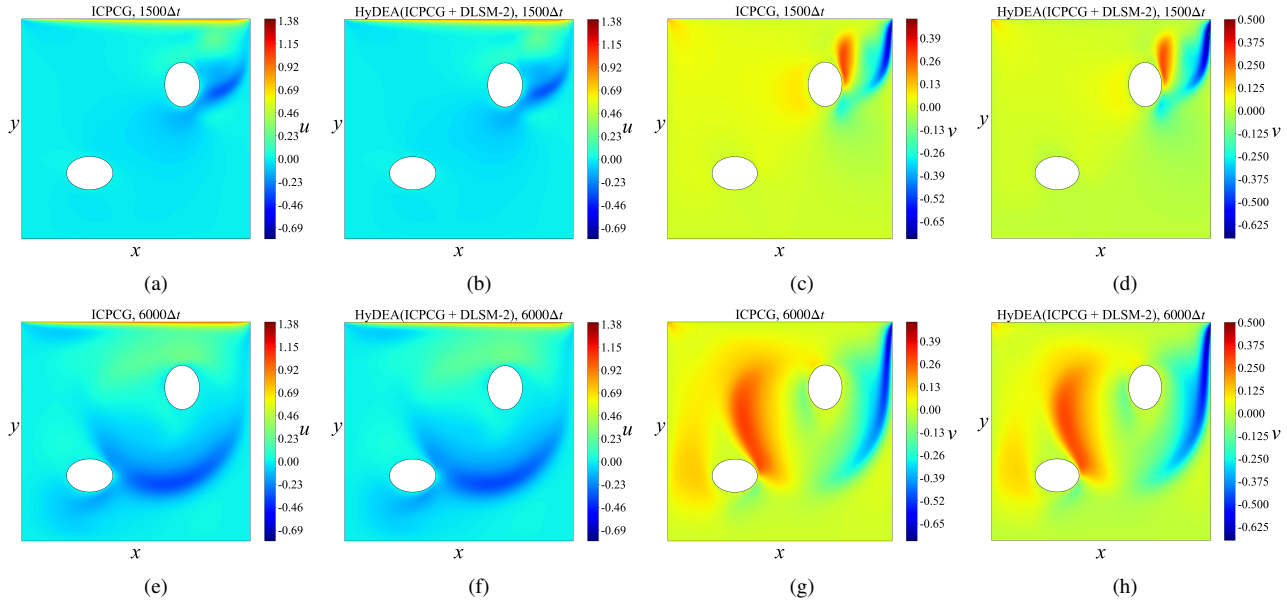


Figure 23: Velocity contours for 2D lid-driven cavity flow with two immersed stationary elliptical cylinders at $Re = 3200$ by ICPCG and HyDEA (ICPCG+DLSM-2). (a)-(d) u and v at the 1500th time step. (e)-(h) u and v at the 6000th time step.

by

$$X = 0.5 - \frac{D \cdot KC}{2\pi} \cdot sin(2\pi ft), \tag{43}$$

$$Y = 0.5, \tag{44}$$

where (X, Y) is its center position and t is time. With frequency $f = 0.5$, cylinder diameter $D = 0.2$ and velocity amplitude $U_m = 0.5$ in x-direction, the Keulegan–Carpenter number $KC = U_m/fD = 5$. Moreover, with kinematic viscosity $\nu = 0.001$, Reynolds number $Re = U_m D/\nu = 100$.

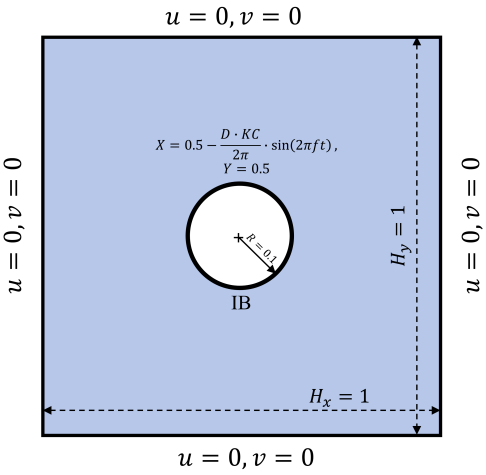


Figure 24: Schematic diagram of 2D flow around an inline oscillating cylinder.

Furthermore, the DIBPM is activated for the oscillating cylinder in CFD computations and the exactly same DLSM-2 as in Section 3.1.2 is executed with $Num_{CG-type} = 3$ and $Num_{DLSM} = 2$, respectively. A detailed analysis for other values of $Num_{CG-type}$ and Num_{DLSM} is also provided in Appendix D.

Taking ICPCG and HyDEA (ICPCG+DLSM-2) as an example, Fig. 25 presents the iterative residuals of solving the PPE at the $10th$, $1000th$ and $3400th$ time steps. For HyDEA (ICPCG+DLSM-2), it takes 10 rounds of the hybrid algorithm with 48 iterations at $10\Delta t$; 4 rounds with 16 iterations at $1000\Delta t$; 4 rounds with 16 iterations at $3400\Delta t$. The results demonstrate that HyDEA achieves a substantial reduction in the number of iterations compared to the ICPCG method alone. Notably, the DeepONet weights employed in this case are identical to those used in Sections 3.1.2, 3.2 and 3.3, which confirms HyDEA's robust generalization capability across diverse flow conditions.

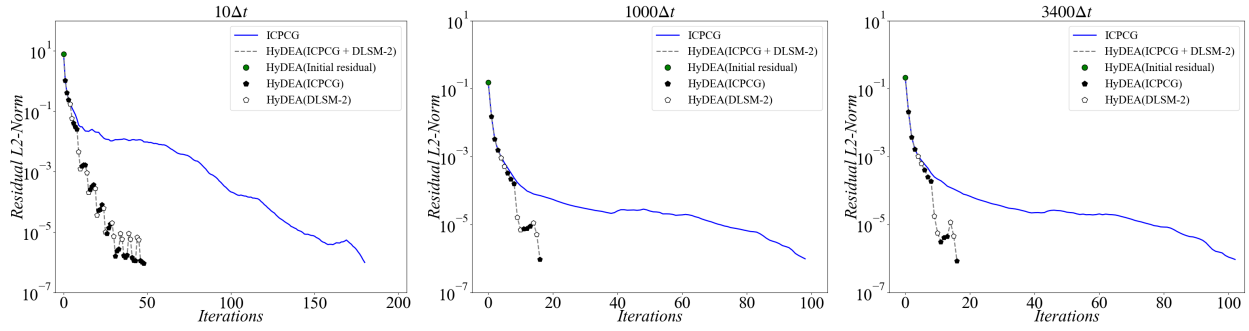


Figure 25: Iterative residuals of solving the PPE for 2D flow around an inline oscillating cylinder at $Re = 100$ and $KC = 5$ by ICPCG and HyDEA (ICPCG+DLSM-2). (a) $10th$ time step. (b) $1000th$ time step. (c) $3400th$ time step.

Fig. 26 presents the velocity contours of ICPCG and HyDEA (ICPCG+DLSM-2) at the $1000th$ and $3400th$ time steps, clearly demonstrating the accurate temporal evolution of the flow fields.

Comparison of DCDM, DLSM and HyDEA

We compare the performance of purely data-driven approaches of DCDM and DLSM with HyDEA and presents the iterative residuals at the $3rd$ and $4th$ time steps, with maximum 400 iterations in Fig. 27. Both DCDM and DLSM show initially effective residual reduction, but their performance degrades significantly for further iterations and cannot reach predefined tolerance of 10^{-6}. Notably, DLSM can sustain the residual at the plateau in the order 10^{-5}, while DCDM even diverges at $t = 4\Delta t$. In contrast, both ICPCG and HyDEA (ICPCG+DLSM-2) can achieve desired convergence, albeit with much less iterations for the latter.

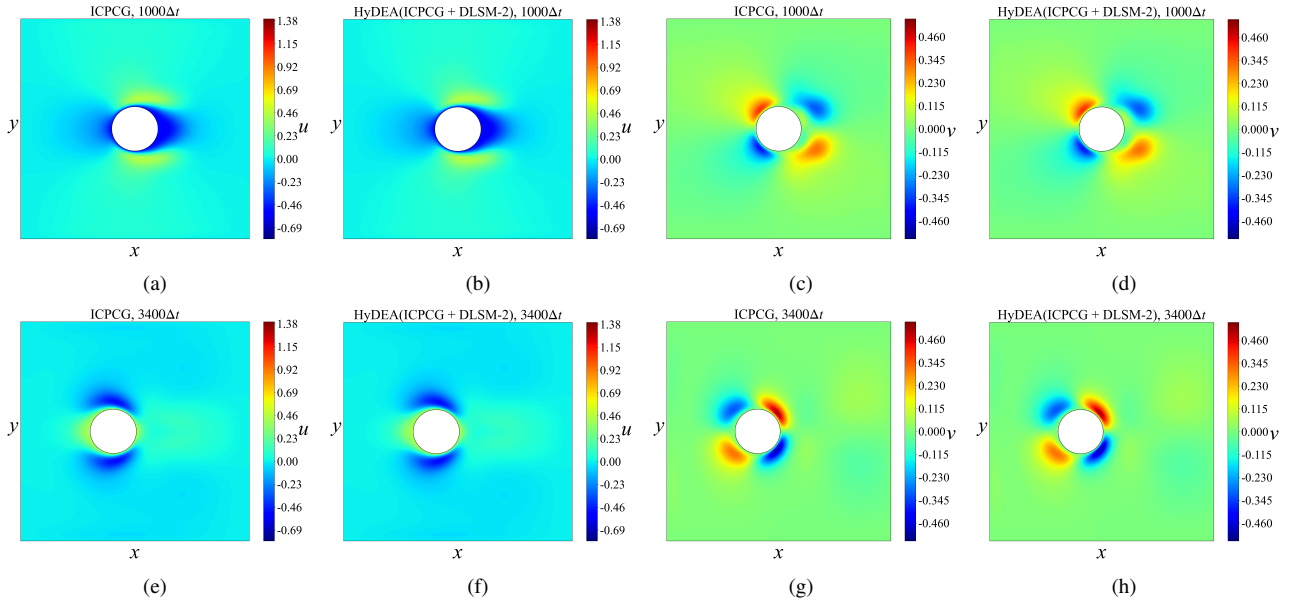


Figure 26: Velocity fields for 2D flow around an inline oscillating cylinder at $Re = 100$ and $KC = 5$ by ICPCG and HyDEA (ICPCG+DLSM-2). (a)-(d) u and v at the 1000th time step. (e)-(h) u and v at the 3400th time step.

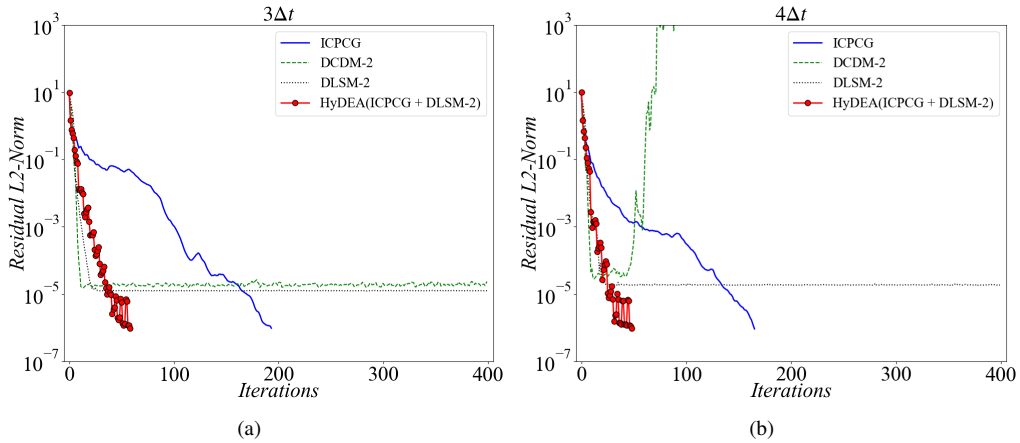


Figure 27: Iterative residuals of solving the PPE for 2D flow around an inline oscillating cylinder at $Re = 100$ and $KC = 5$ with resolution 192×192: comparisons among ICPCG, data-driven approaches of DCDM and DLSM, and HyDEA. (a) 3rd time step. (b) 4th time step.

Comparison of two versions of HyDEA

As indicated in Fig. 2, HyDEA offers two distinct implementations shown in Fig. 2(b) and 2(c) for each round of the hybrid algorithm. We examplify the comparison of the two versions for the same scenario, namely HyDEA-I (ICPCG+DLSM-2) versus HyDEA-II (DLSM-2+ICPCG) for:

- CFD resolution of 192×192;
- $Num_{CG-type} = 3$ and $Num_{DLSM} = 2$;
- maximum 400 iteration at one time step.

Fig. 28 presents the iterative residuals of solving the PPE using ICPCG, HyDEA-I and HyDEA-II at the $10th$ and $100th$ time steps. The results reveal a clear performance contrast, with HyDEA-I exhibiting superior convergence characteristics compared to that of HyDEA-II. The observed performance superiority stems from the fact that in HyDEA-I, a CG-type method initially eliminates high-frequency errors to a certain extent before passing onto the DLSM module. Subsequently, DLSM/DeepONet receives an better input, which aligns with the training dataset, and is able to predict more sensible line-search directions, ultimately leading to a faster convergence. Based on these discussions and demonstrated gains in performance, HyDEA-I emerges as the recommended version, which was also extensively employed in previous benchmark cases in Sections 3.1, 3.2, 3.3, and 3.4.

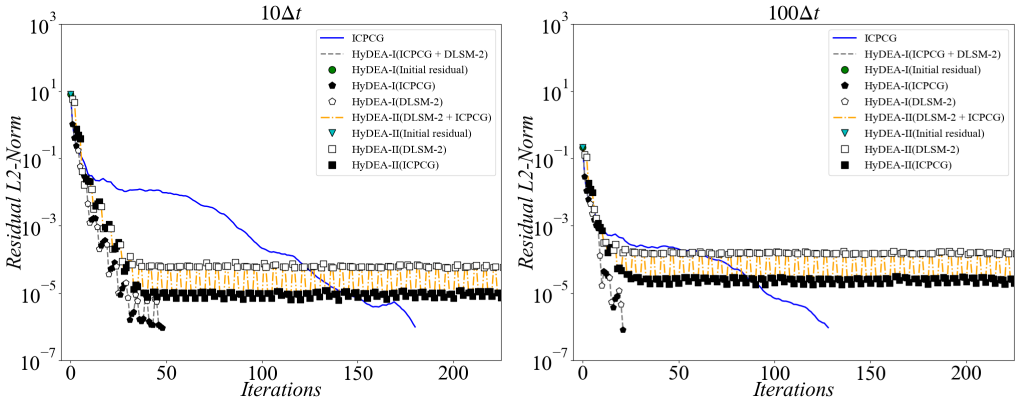


Figure 28: Iterative residuals of solving the PPE for 2D flow around an inline oscillating cylinder at $Re = 100$ and $KC = 5$ by two different implementations of HyDEA . (a) $10th$ time step. (b) $100th$ time step.

Comparison of DLSM and DCDM as module in HyDEA

The DCDM of Kaneda et al. [32] represents a special case of DLSM, where the line-search directions are made additionally M-orthogonal by a GS orthogonalization process. We evaluate respective performance of DLSM module and DCDM module within the HyDEA framework. We compare iterative residuals of HyDEA (ICPCG+DLSM-2) and HyDEA (ICPCG+DCDM-2) for solving the PPE at the $10th$, $1000th$ and $3400th$ time steps in Fig. 29. The results show that HyDEA achieves similar convergence rates when equipped with either DLSM-2 or DCDM-2, suggesting that GS orthogonalization of DeepONet-predicted line-search directions provides negligible benefits.

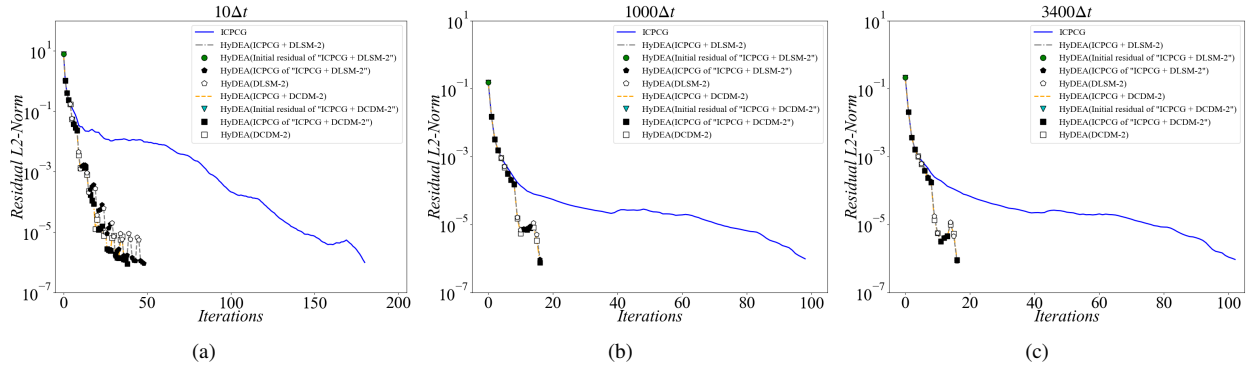


Figure 29: Iterative residuals of solving the PPE for 2D flow around an inline oscillating cylinder at $Re = 100$ and $KC = 5$ by ICPCG, HyDEA (ICPCG+DLSM-2), and HyDEA (ICPCG+DCDM-2). (a) $10th$ time step. (b) $1000th$ time step. (c) $3400th$ time step.

4. Conclusion

We present HyDEA, a novel hybrid framework that synergizes deep learning with classical iterative solvers to accelerate incompressible flow simulations. By targeting the pressure Poisson equation (PPE) - the computational bottleneck in the fractional step method - HyDEA combines the strengths of a Deep Learning line-search method (DLSM) and a conjugate gradient (CG)-type solver. The DLSM module, powered by DeepONet, predicts optimal line-search directions to accelerate convergence for the large, sparse, symmetric positive-definite linear systems arising from finite difference discretizations. For flows with solid structures, the decoupled immersed boundary projection method seamlessly integrates into HyDEA while preserving the linear system's structure. HyDEA addresses the limitations of purely data-driven approaches by leveraging complementary strengths:

- DeepONet captures efficiently large-scale solution features.
- CG-type methods (including preconditioned variants with Incomplete Cholesky, Jacobi, and Multigrid) robustly resolve high-frequency errors.

Benchmark results demonstrate HyDEA's superior performance and generalization:

- Reduces iteration counts significantly compared to standalone CG-type methods across diverse flows (single-component, stationary/moving obstacles) using *fixed network weights*.
- Exhibits super-resolution capability, achieving accurate solutions on a 512×512 grid despite being trained on 128×128 data ($16\times$ resolution increase).
- Outperforms standalone DLSM and purely data-driven methods (e.g., DCDM), proving neural networks augment but cannot replace classical solvers without sacrificing stability or accuracy.

The prototype implementation of HyDEA combines heterogeneous frameworks: `Pytorch` (`Python`) for DLSM module and `PETSc` (C) for CG-type solvers, inducing GPU-CPU communication overhead. Additionally, the CNN-based DeepONet restricts applications to uniform Cartesian grids. Future efforts will focus on code optimization by minimizing data transfer latency between frameworks and extending HyDEA to non-uniform grids for broader applicability. HyDEA's hybrid paradigm bridges the gap between data-driven efficiency and numerical robustness, offering a scalable pathway for high-fidelity flow simulations.

Acknowledgments

The authors appreciate support from National Key R&D Program of China (2022YFA1203200).

Appendix A. Analysis of convergence behavior of the CG method

This section analyzes the convergence behavior of the CG method, aiming to provide a rationale for the dataset construction method in Section 2.4.1. The CG method is an efficient algorithm for solving the quadratic vector optimization problem formulated in Eq. (35). By substituting $\delta p = \delta p^{true} - e$, where e denotes the iterative error vector, we can reformulate Eq. (35) as:

$$\min_{e\in\mathbf{R}^a}\{\frac{1}{2}e^T Me - \frac{1}{2}(\delta p^{true})^T M\delta p^{true} + c\}. \tag{A.1}$$

Since the second and third terms in Eq. (A.1) are constant, the optimization problem reduces to:

$$\min_{e\in\mathbf{R}^a}\{e^T Me = \|e\|_M^2\}, \tag{A.2}$$

where $\|e\|_M = \sqrt{e^T Me}$ denotes the energy norm of the error. The CG method progressively minimizes $\|e\|_M$ during its iterative solution process [35].

The normalized eigenvectors $\{v_0, v_1, \cdots, v_{a-1}\}$ of the symmetric-positive-definite matrix M form an orthonormal basis, enabling the initial error vector e_0 to be expressed as a linear combination of these normalized eigenvectors:

$$e_0 = \sum_{j=0}^{a-1} \alpha_j v_j, \tag{A.3}$$

where coefficient α_j represents the projection length of e_0 along the direction of v_j.

The error vector at the kth iteration can be expressed in the following form [43]:

$$e_k = P_k(M)e_0, \tag{A.4}$$

where P_k is a k-degree polynomial satisfying $P_k(0) = 1$. By substituting Eqs. (A.3) and (A.4) into the $\|e\|_M = \sqrt{e^T M e}$, the discrete formulation for $\|e\|_M^2$ at the kth iteration is obtained as follows:

$$\|e_k\|_M^2 = \sum_{j=0}^{a-1} \alpha_j^2 [P_k(\lambda_j)]^2 \lambda_j, \tag{A.5}$$

where λ_j is the eigenvalue corresponding to v_j. The CG method seeks an optimal polynomial P_k that minimizes Eq. (A.5). Furthermore, to analyze the convergence behavior, the upper bound for $\|e_k\|_M^2$ can be derived as follows:

$$\|e_k\|_M^2 \leq \min_{P_k} \max_{\lambda \in \Lambda(M)} [P_k(\lambda)]^2 \sum_{j=0}^{a-1} \alpha_j^2 \lambda_j = \min_{P_k} \max_{\lambda \in \Lambda(M)} [P_k(\lambda)]^2 \|e_0\|_M^2. \tag{A.6}$$

where $\Lambda(M)$ denotes the set of eigenvalues of M.

Remarkably, $\max_{\lambda \in \Lambda(M)} [P_k(\lambda)]^2$ is minimized when P_k adopts the following form [43, 58]:

$$P_k(\lambda) = \frac{Q_k(\frac{\lambda_{max} + \lambda_{min} - 2\lambda}{\lambda_{max} - \lambda_{min}})}{Q_k(\frac{\lambda_{max} + \lambda_{min}}{\lambda_{max} - \lambda_{min}})}, \tag{A.7}$$

where λ_{max} and λ_{min} correspond to the largest and smallest eigenvalues of M, respectively. The polynomial Q_k denotes the Chebyshev polynomial of degree k. Essentially, P_k is constructed through a scaling transformation of Q_k. This enables analysis of the properties of P_k through the well-established characteristics of Q_k. Four representative Chebyshev polynomials of different degrees are illustrated in Fig. A.30.

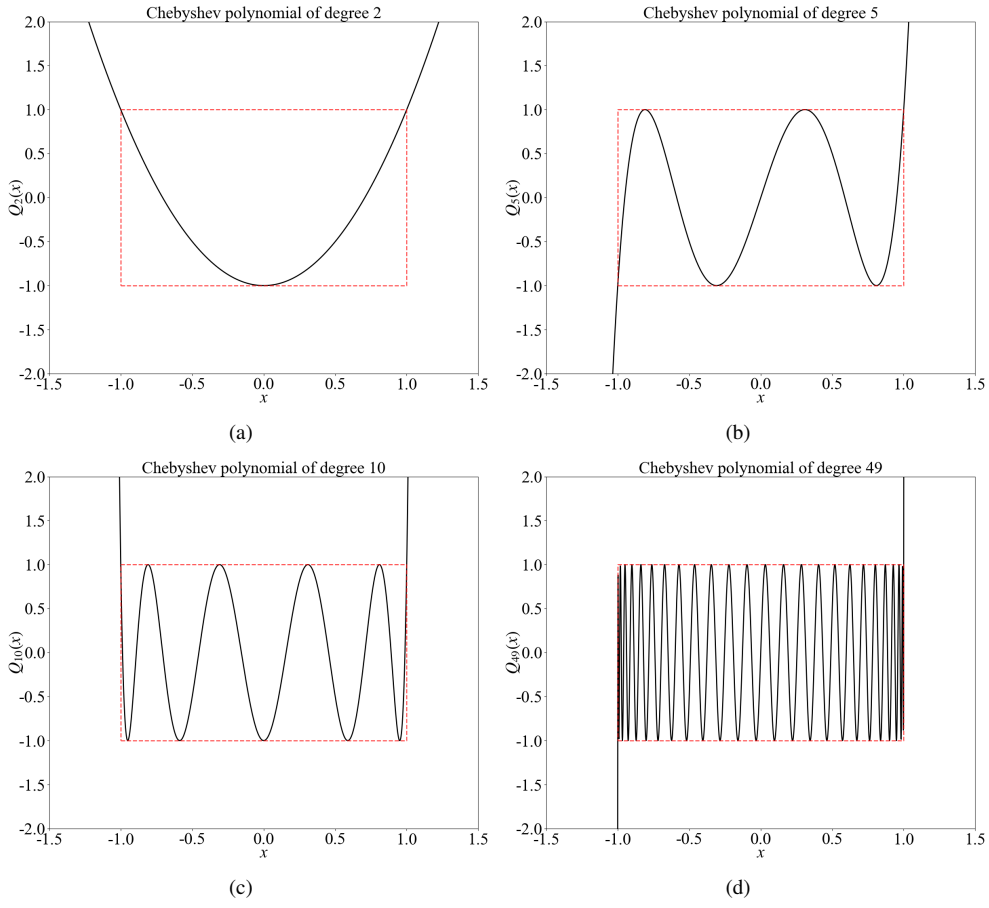
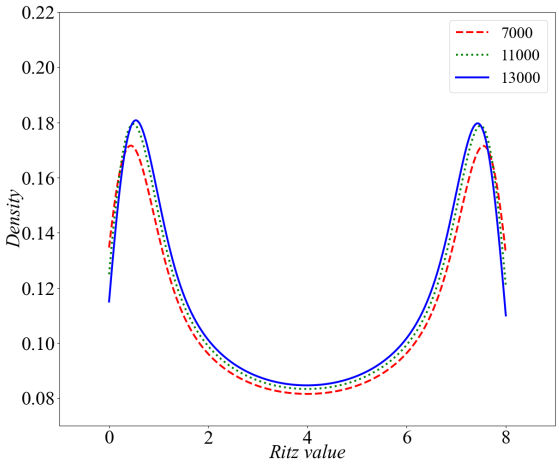


Figure A.30: Four representative Chebyshev polynomials of different degrees. (a) Degree 2. (b) Degree 5. (c) Degree 10. (d) Degree 49.

Analysis of Eq. (A.7) reveals that P_k exhibits increasingly rapid oscillations with a higher zero point density as λ approaches the spectral boundary regions near λ_{max} and λ_{min}. Consequently, the eigenvalue distribution of M exerts a fundamental influence on the convergence behavior of the CG method.

The Ritz values derived from the Lanczos iteration approximate the eigenvalue distribution of M, with extreme eigenvalues converging rapidly during the early iterations [50]. Based on the uniform grid configuration (detailed in Section 3.1.2 with grid resolution of 192×192), we calculate the Ritz values at three distinct Lanczos iteration numbers (7, 000,



(a)

Figure A.31: Probability density functions of Ritz values at distinct Lanczos iteration counts.

11,000, and 13,000), with the corresponding probability density function (PDF) curves presented in Fig. A.31. The results show that the eigenvalues of M are predominantly concentrated in the spectral boundary regions.

Furthermore, we assume that the initial error e_0 is spectrally unbiased (i.e. coefficients α_j in Eq. (A.3) are approximately equal across all indexes j). By analyzing:

- the properties of P_k (Eq. (A.7)),
- the PDF curves of Ritz values (Fig. A.31), and
- the discrete formulation of $\|e_k\|_M^2$ (Eq. (A.5)),

we can conclude that the components of $\|e_k\|_M^2$ corresponding to the larger eigenvalues of M decay more rapidly during the CG iterations, because larger eigenvalues enhance the damping effect of P_k. In contrast, the components of $\|e_k\|_M^2$ associated with smaller eigenvalues require more CG iterations to achieve significant reduction.

Following the formulation of Eq. (A.5), the iteration residual vector at kth iteration can be expressed in the following discrete form:

$$r_k = Me_k = \sum_{j=0}^{a-1} \alpha_j P_k(\lambda_j)\lambda_j v_j. \tag{A.8}$$

Being consistent with the established analysis, the components of r_k associated with the larger eigenvalues of M decay more rapidly during the CG iterations.

In light of the above analysis, the dataset construction method presented in Section 2.4.1 provides a theoretically justified framework under the spectrally unbiased assumption of e_0. However, in practical applications to solve the PPE central to the fractional step method in incompressible flow simulations, the spectral properties of e_0 are intrinsically problem dependent and generally unknown a priori. This inherent uncertainty necessitates a parametric analysis of the key hyper-parameters in dataset construction.

Appendix B. Sensitivity analysis of neural network parameters A, B, C and T on HyDEA's performance

This section analyzes the impact of neuron numbers A, B, and C and channel number T on HyDEA's performance. The analysis is conducted based on the flow configuration described in Section 3.1.1, and more specifically:

- 2D lid-driven cavity flow at $Re = 1000$ using grid resolution of 128×128, with $b = 0.6$, $m = 3000$, $Num_{CG-type} = 3$ and $Num_{DLSM} = 2$.

To avoid over-extensive comparisons, the HyDEA (CG+DLSM-1) and HyDEA (ICPCG+DLSM-1) are employed in the forecasting stage.

Sensitivity analysis of neuron numbers A, B, C

We systematically analyze the impact of neuron numbers A, B, and C on HyDEA's performance, with the channel number $T = 40$. The tested configurations summarized in Table B.4.

Table B.4: The values of neuron number in the FNN component of the DeepONet

Neuron number	Case I	Case II	Case III	Case IV	Case V
A	50	75	100	150	250
B	100	150	200	300	500
C	50	75	100	150	250

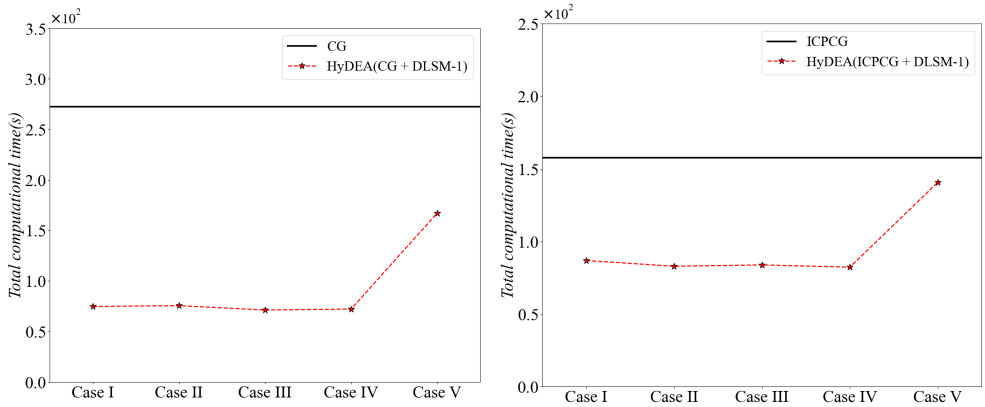


Figure B.32: Comparison of total computational time required to solve the PPE across Cases I to V. (a) HyDEA (CG+DLSM-1). (b) HyDEA (ICPCG+DLSM-1).

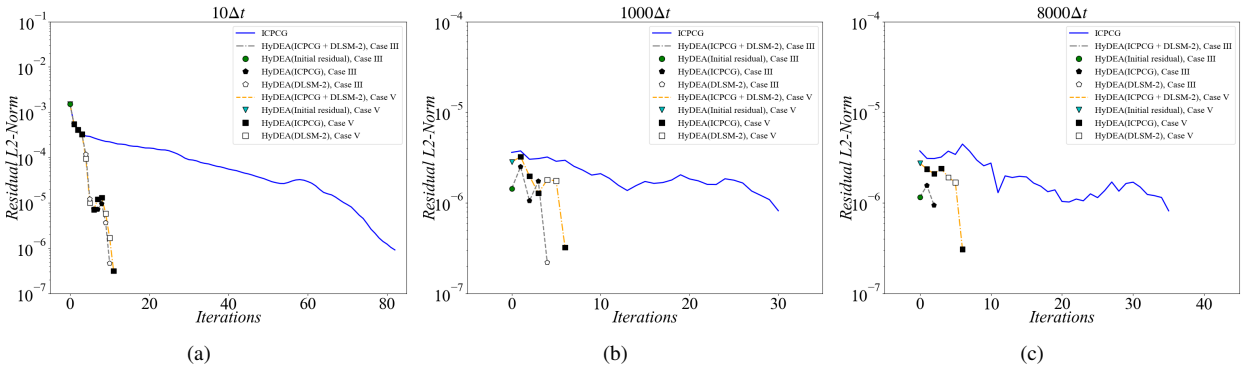


Figure B.33: The iterative residuals of solving the PPE for Cases III and V. (a) 10*th* time step. (b) 1000*th* time step. (c) 8000*th* time step.

Figs. B.32 presents the total computational time over 10,000 consecutive time steps required to solve the PPE for Cases I to V, and the iterative residuals of solving the PPE using HyDEA (ICPCG+DLSM-1) for Cases III and V at three representative time steps are depicted in Fig. B.33.

The computational times for Cases I to IV are comparable, while Case V demonstrates notably inferior performance. This discrepancy likely stems from the increased number of network parameters, which hinders convergence to optimal network parameters within the limited number of training epochs. This limitation forces HyDEA to perform additional iterations to achieve the prescribed residual threshold, as demonstrated by the convergence curves in Fig. B.33. Specifically, for case III, it takes 2 rounds with 10 iterations at $10\Delta t$, 1 round with 4 iterations at $100\Delta t$, and 1 round with 2 iterations at $8000\Delta t$; for case V, it takes 3 rounds with 11 iterations at $10\Delta t$, 2 rounds with 6 iterations at $100\Delta t$, and 2 rounds with 6 iterations at $8000\Delta t$.

Although additional training epochs may improve results for Case V, this would also induce higher computational cost. It is particularly noteworthy that HyDEA requires significantly fewer iterations than the ICPCG method to solve the PPE for both Case III and Case V, a result that warrants special emphasis.

Sensitivity analysis of channel number T

Subsequently, we analyze the impact of channel number T by evaluating three configurations: $T = 20$, $T = 30$ and $T = 40$. For all cases, we maintain identical neuron numbers fixed at $A = 100$, $B = 200$, and $C = 100$. A comparison of the total computational time required to solve the PPE over 10,000 consecutive time steps for these three cases is illustrated in Fig. B.34.

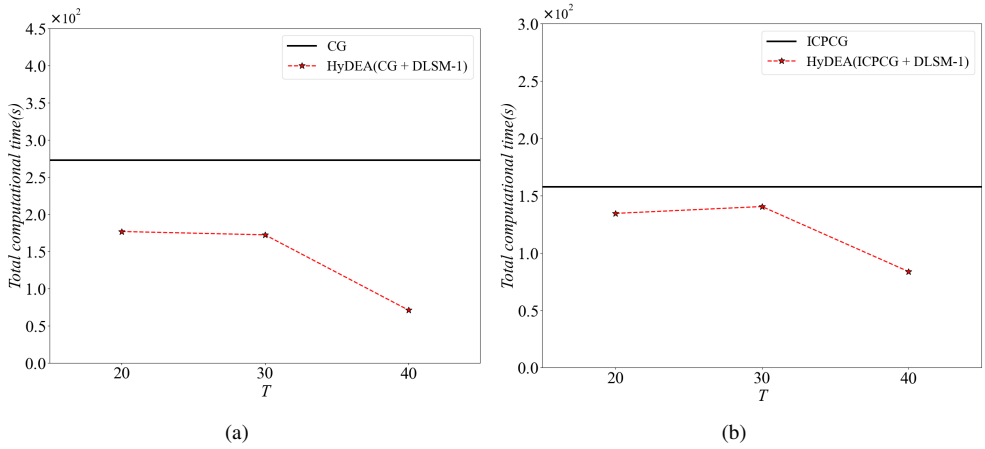


Figure B.34: Comparison of total computational time required to solve the PPE across varying channel number T. (a) HyDEA (CG+DLSM-1). (b) HyDEA (ICPCG+DLSM-1).

The results demonstrate that cases with $T = 20$ and 30 require higher computational time compared to $T = 40$. The inferior performance of DeepONet at these lower channel numbers suggests that reduced feature space dimensionality compromises the model's ability to accurately learn the functional mapping between e_k and r_k spaces. This limitation forces HyDEA to perform additional iterations to achieve the prescribed residual threshold, as evidenced by the iterative residuals of $T = 20$ and 40 in Fig. B.35, ultimately increasing total computational time. Specifically, for $T = 20$, it takes 3 rounds with 11 iterations at $10\Delta t$, 2 rounds with 6 iterations at $100\Delta t$, and 2 rounds with 6 iterations at $8000\Delta t$; for $T = 40$, it takes 2 rounds with 10 iterations at $10\Delta t$, 1 round with 4 iterations at $100\Delta t$, and 1 round with 2 iterations at $8000\Delta t$.

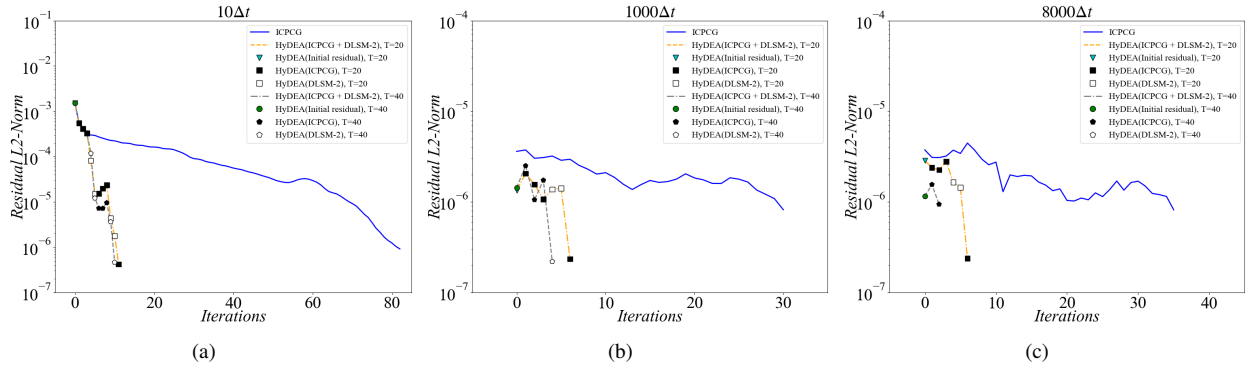


Figure B.35: The iterative residuals of solving the PPE for $T = 20$ and $T = 40$. (a) $10th$ time step. (b) $1000th$ time step. (c) $8000th$ time step.

Appendix C. Sensitivity analysis of dataset construction parameters m and b on HyDEA's performance

This section conducts a systematic sensitivity analysis to quantify the influence of dataset construction parameters m and b on HyDEA's performance. The analysis is conducted based on the 2D lid-driven cavity flows described in Sections 3.1.1 and 3.1.2, which employ two grid resolutions. More specifically:

- $Re = 1000$ using grid resolution of 128×128, with $Num_{CG-type} = 3$ and $Num_{DLSM} = 2$.
- $Re = 3200$ using grid resolution of 192×192, with $Num_{CG-type} = 3$ and $Num_{DLSM} = 2$.

2D lid-driven cavity flow at $Re = 1000$

We first analyze the configuration of Section 3.1.1. The specific values of m and b are summarized in Table C.5. The combination of the two parameters generates 15 groups of the training dataset.

Table C.5: The values of dataset construction parameters m and b in the case of Section 3.1.1

Parameter	Value
m	2000, 3000, 5000, 7000, 9000
b	0.4, 0.5, 0.6

Fig. C.36 presents a comprehensive comparison of total computational time between HyDEA (CG-type+DLSM-1) and CG-type methods required to solve the PPE over 10,000 consecutive time steps under varying dataset construction parameters.

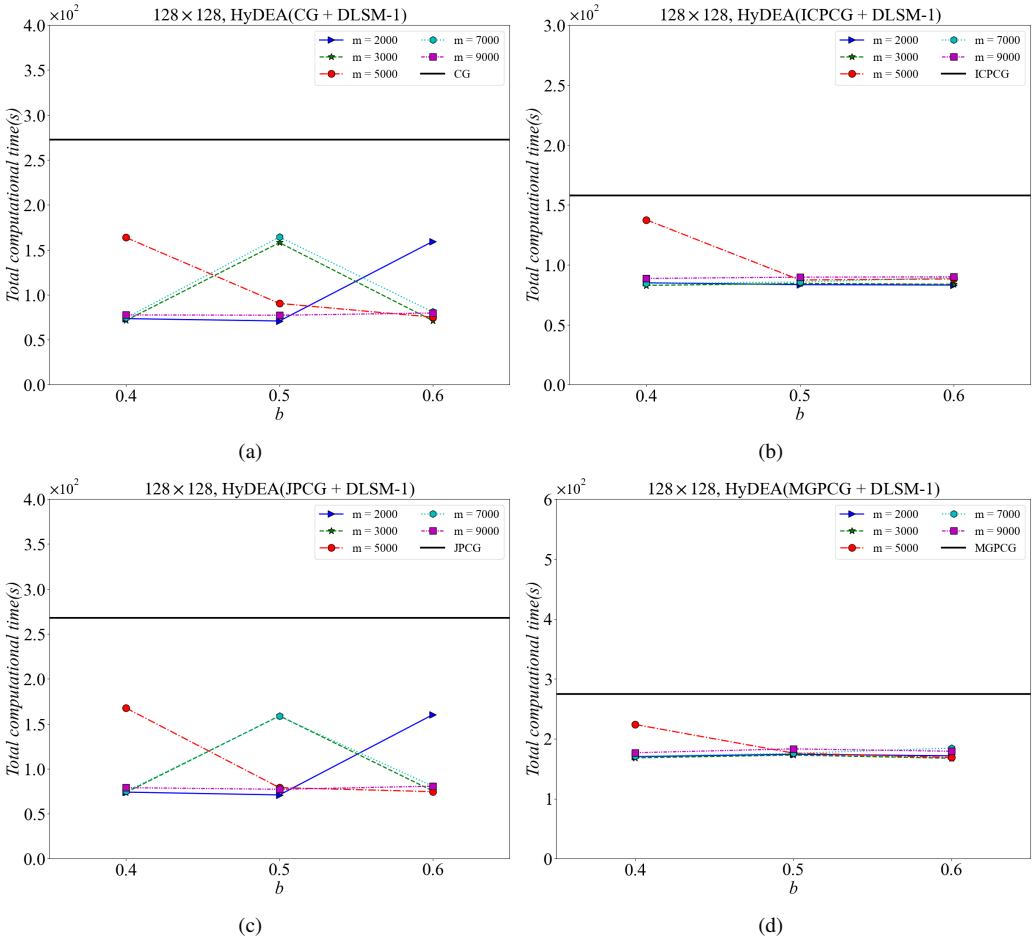


Figure C.36: Comparison of total computational time required to solve the PPE for 2D lid-driven cavity at $Re = 1000$: influence of different values of m and b. (a) HyDEA (CG+DLSM-1). (b) HyDEA (ICPCG+DLSM-1). (c) HyDEA (JPCG+DLSM-1). (d) HyDEA (MGPCG+DLSM-1).

With appropriate parameter selection, HyDEA exhibits lower computational time compared to CG-type methods. Notably, when $b = 0.6$, the results of HyDEA demonstrate enhanced performance in most values of m.

2D lid-driven cavity flow at Re = 3200

Furthermore, for the configuration in Section 3.1.2, the specific values of m and b are summarized in Table C.6, and a comparison of the total computational time over 10,000 consecutive time steps required to solve the PPE is illustrated in Fig. C.37.

Table C.6: The values of dataset construction parameters m and b in the case of Section 3.1.2

Parameter	Value
m	3000, 5000, 7000, 9000, 11000
b	0.4, 0.5, 0.6

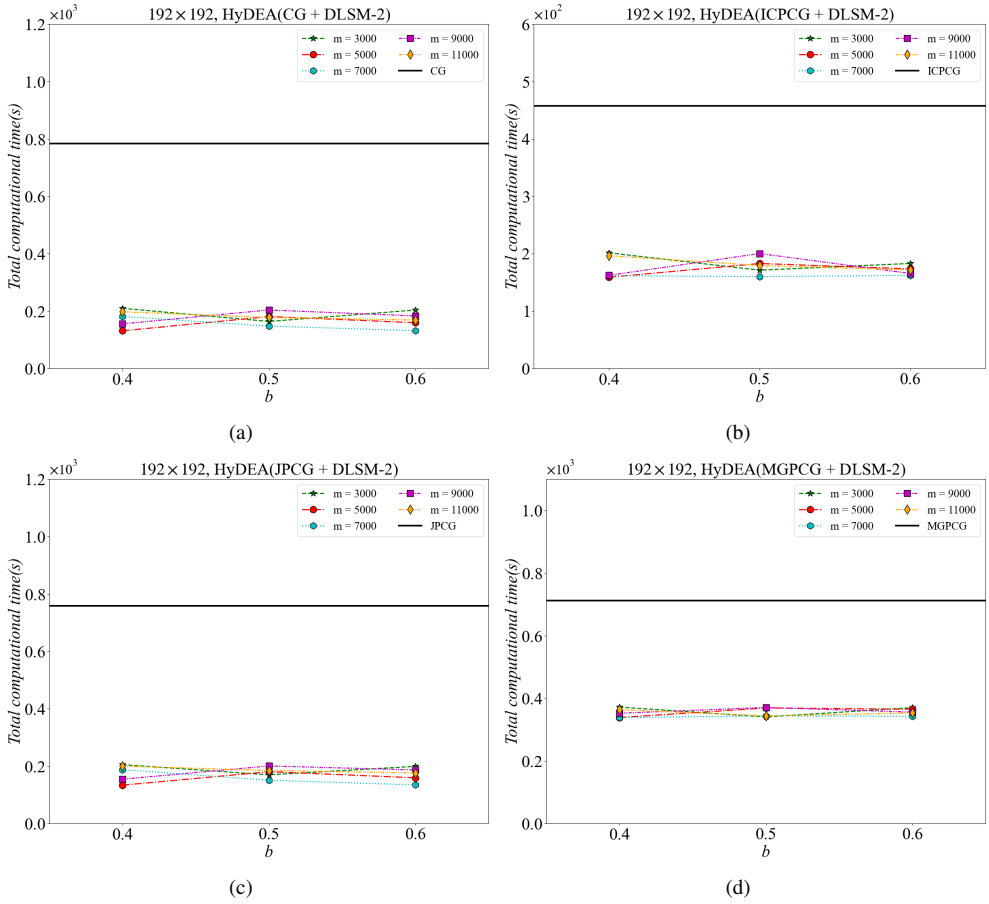


Figure C.37: Comparison of total computational time required to solve the PPE for 2D lid-driven cavity at $Re = 3200$: influence of different values of b and m. (a) HyDEA (CG+DLSM-2). (b) HyDEA (ICPCG+DLSM-2). (c) HyDEA (JPCG+DLSM-2). (d) HyDEA (MGPCG+DLSM-2).

The results demonstrate that HyDEA consistently outperforms CG-type methods in computational efficiency across all tested parameter combinations, and shows particularly more stable time costs compared to low-resolution scenarios. This robust performance advantage underscores HyDEA's enhanced adaptability to high-resolution simulations, as evidenced by its greater tolerance to variations in dataset construction parameters.

Appendix D. Sensitivity analysis of $Num_{CG-type}$ and Num_{DLSM} on HyDEA's performance

This section presents a systematic sensitivity analysis to quantify the influence of key computational parameters $Num_{CG-type}$ and Num_{DLSM} on HyDEA's performance. The analysis is conducted based on two flow configurations described in Sections 3.1.1 and 3.4, and more specifically:

- 2D lid-driven cavity flow at $Re = 1000$ with grid resolution of 128×128, which evolves toward a steady state. The dataset is constructed using $m = 3000$ and $b = 0.6$.
- 2D flow around an inline oscillating cylinder at $Re = 100$ with grid resolution of 192×192, which maintains inherently unsteady characteristics. The dataset is constructed using $m = 7000$ and $b = 0.6$.

2D lid-driven cavity flow at Re = 1000

We first analyze the configuration of Section 3.1.1, holding Num_{DLSM} constant at 2 while varying $Num_{CG-type}$ from 1 to 5 to investigate its impact.

Taking HyDEA (CG+DLSM-1) and HyDEA (ICPCG+DLSM-1) as representative cases, Fig. D.38 presents the total computational time required to the PPE over 10,000 consecutive time steps. The results demonstrate that the computational time exhibits a consistent trend across varying values of $Num_{CG-type}$ in this case.

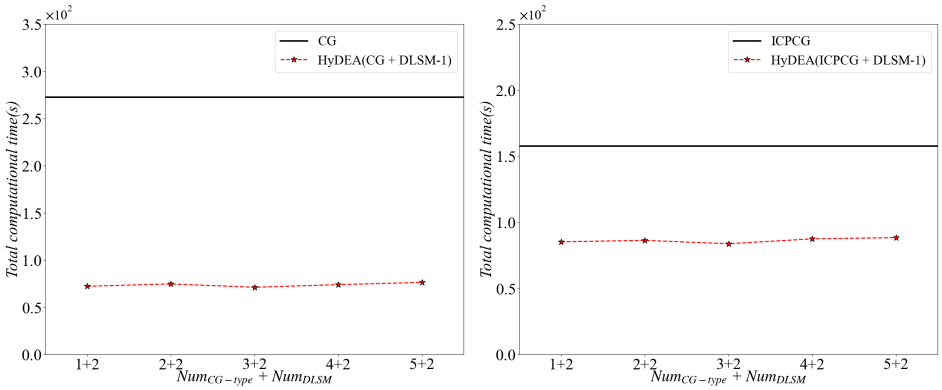


Figure D.38: Comparison of total computational time required to solve PPE across varying values of $Num_{CG-type}$ for 2D lid-driven cavity flow at $Re = 1000$. (a) HyDEA (CG+DLSM-1). (b) HyDEA (ICPCG+DLSM-1).

Furthermore, $Num_{CG-type}$ is held constant at 3 while varying Num_{DLSM} from 1 to 5 to investigate its impact. The total computational time required to solve the PPE over 10,000 consecutive time steps is presented in Fig. D.39. The results demonstrate that varying Num_{DLSM} has a negligible impact on the total computational time in this case. This insensitivity occurs because the flow field approaches the steady state over time, the DLSM module typically meets the residual threshold before reaching the specified maximum Num_{DLSM} iteration limit for the PPE solution.

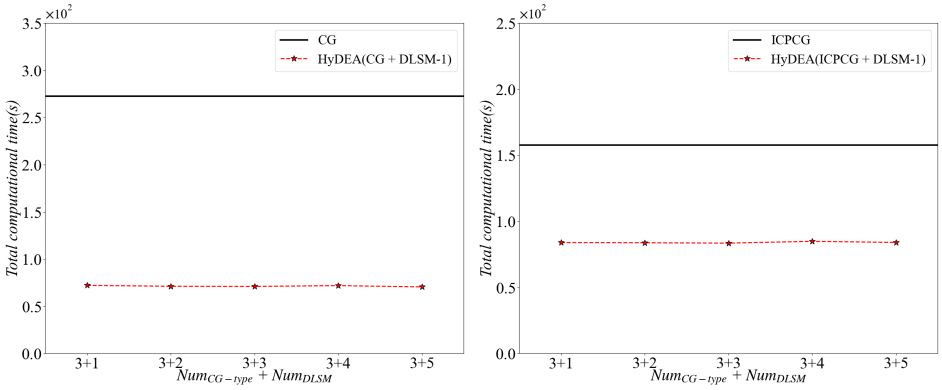


Figure D.39: Comparison of computational time for different values of Num_{DLSM} for 2D lid-driven cavity flow at $Re = 1000$. (a) HyDEA (CG+DLSM-1). (b) HyDEA (ICPCG+DLSM-1).

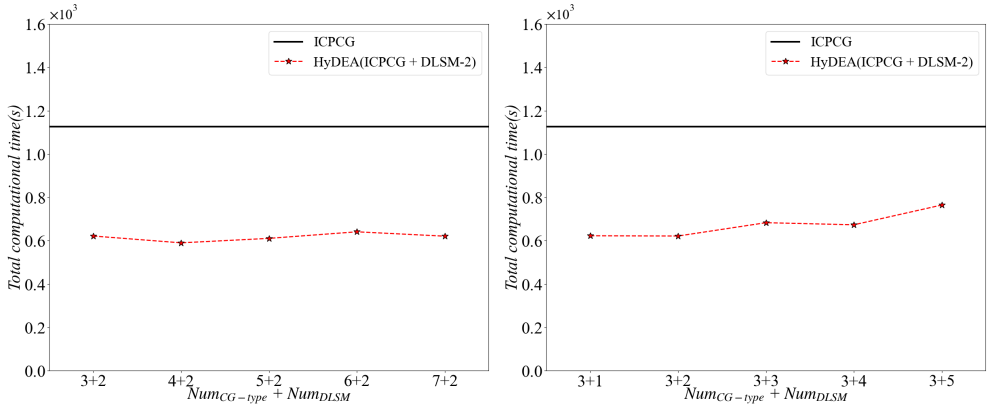
2D flow around an inline oscillating cylinder at Re = 100 and KC = 5

Subsequently, we perform a parametric analysis to evaluate the influence of $Num_{CG-type}$ and Num_{DLSM} in the case presented in Section 3.4.

The Num_{DLSM} is fixed at 2 while varying $Num_{CG-type}$ from 1 to 7 to investigate its impact, with the result presented in Fig. 40(a). Notably, we exclude the results of $Num_{CG-type} = 1$ and 2 from the analysis due to their unsatisfactory performance. For values of $Num_{CG-type}$ above 2, the computational efficiency are comparable.

Furthermore, the $Num_{CG-type}$ is initially fixed at 3 while varying Num_{DLSM} from 1 to 5 to investigate its impact. Fig. 40(b) presents the total computational time required to solve the PPE over 10,000 consecutive time steps. The results demonstrate that increasing Num_{DLSM} leads to a slight reduction in overall computational efficiency. This degradation occurs because more frequent consecutive DLSM invocations during the computation. Interestingly, this trend contrasts with the observations in Fig. D.39, because the lid-driven cavity flow will gradually reaches steady-state conditions.

Based on the above results, we recommend set $Num_{CG-type}$ to be larger than Num_{DLSM} in HyDEA for complex flow simulations to maintain optimal computational performance.



(a) (b)

Figure D.40: Comparison of total computational time required to solve the PPE across varying values of $Num_{CG-type}$ and Num_{DLSM} for 2D flow around an inline oscillating cylinder. (a) $Num_{DLSM} = 2$, $Num_{CG-type}$ varies across the values $3, 4, 5, 6$ and 7. (b) $Num_{CG_type} = 2$, Num_{DLSM} varies across the values $1, 2, 3, 4$ and 5.

References

[1] J. F. Wendt (Ed.), Computational Fluid Dynamics: An introduction, 3rd Edition, Springer-Verlag Berlin Heidelberg, 2009.

[2] J. Rabault, M. Kuchta, A. Jensen, U. Réglade, N. Cerardi, Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control, Journal of Fluid Mechanics 865 (2019) 281–302.

[3] J. Viquerat, J. Rabault, A. Kuhnle, H. Ghraieb, A. Larcher, E. Hachem, Direct shape optimization through deep reinforcement learning, Journal of Computational Physics 428 (2021) 110080.

[4] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[5] R. Zhang, H. Lee, D. Radev, Dependency sensitive convolutional neural networks for modeling sentences and documents, arXiv preprint arXiv:1611.02361 (2016).

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

[7] S. Cai, Z. Mao, Z. Wang, M. Yin, G. E. Karniadakis, Physics-informed neural networks (PINNs) for fluid mechanics: a review, Acta Mechanica Sinica 37 (12) (2021) 1727–1738.

[8] P.-Y. Chuang, L. A. Barba, Experience report of physics-informed neural networks in fluid simulations: pitfalls and frustration, arXiv preprint arXiv:2205.14249 (2022).

[9] P. Karnakov, S. Litvinov, P. Koumoutsakos, Solving inverse problems in physics by optimizing a discrete loss: Fast and accurate learning without neural networks, PNAS nexus 3 (1) (2024) 1–16.

[10] Y. Zhu, W. Kong, J. Deng, X. Bian, Physics-informed neural networks for incompressible flows with moving boundaries, Physics of Fluids 36 (2024) 013617.

[11] S. Wang, X. Yu, P. Perdikaris, When and why pinns fail to train: A neural tangent kernel perspective, Journal of Computational Physics 449 (2022) 110768.

[12] J. Yu, L. Lu, X. Meng, G. E. Karniadakis, Gradient-enhanced physics-informed neural networks for forward and inverse pde problems, Computer Methods in Applied Mechanics and Engineering 393 (2022) 114823.

[13] S. Lee, D. You, Data-driven prediction of unsteady flow over a circular cylinder using deep learning, Journal of Fluid Mechanics 879 (2019) 217–254.

[14] T. Nakamura, K. Fukami, K. Hasegawa, Y. Nabae, K. Fukagata, Convolutional neural network and long short-term memory based reduced order surrogate for minimal turbulent channel flow, Physics of Fluids 33 (2021) 025116.

[15] T. Chen, H. Chen, Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems, IEEE Transactions on Neural Networks 6 (4) (1995) 911–917.

[16] L. Lu, P. Jin, G. Pang, Z. Zhang, G. E. Karniadakis, Learning nonlinear operators via deeponet based on the universal approximation theorem of operators, Nature Machine Intelligence 3 (3) (2021) 218–229.

[17] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, arXiv preprint arXiv:2010.08895 (2020).

[18] C. Lin, M. Maxey, Z. Li, G. E. Karniadakis, A seamless multiscale operator neural network for inferring bubble dynamics, Journal of Fluid Mechanics 929 (2021) A18.

[19] G. Wen, Z. Li, K. Azizzadenesheli, A. Anandkumar, S. M. Benson, U-FNO—An enhanced Fourier neural operator-based deep-learning model for multiphase flow, Advances in Water Resources 163 (2022) 104180.

[20] Z. Mao, L. Lu, O. Marxen, T. A. Zaki, G. E. Karniadakis, DeepM&Mnet for hypersonics: Predicting the coupled flow and finite-rate chemistry behind a normal shock using neural-network approximation of operators, Journal of Computational Physics 447 (2021) 110698.

[21] H. Bai, Z. Wang, X. Chu, J. Deng, X. Bian, Data-driven modeling of unsteady flow based on deep operator network, Physics of Fluids 36 (2024) 063610.

[22] R. Han, X. Yang, W. Liu, Unsteady flow prediction based on a hybrid network-operator learning model, Physics of Fluids 37 (2) (2025) 027150.

[23] X. Zhang, F. Xie, T. Ji, C. Zheng, H. Zheng, Y. Zheng, Parametric unsteady flow modeling by using meta learning, Engineering Applications of Artificial Intelligence 121 (2023) 105978.

[24] B. Xu, Y. Zhou, X. Bian, Self-supervised learning based on transformer for flow reconstruction and prediction, Physics of Fluids 36 (2024) 023607.

[25] S. Goswami, K. Kontolati, M. D. Shields, G. E. Karniadakis, Deep transfer operator learning for partial differential equations under conditional shift, Nature Machine Intelligence 4 (2022) 1155–1164.

[26] C. Yang, X. Yang, X. Xiao, Data-driven projection method in fluid simulation, Computer Animation and Virtual Worlds 27 (3-4) (2016) 415–424.

[27] J. Tompson, K. Schlachter, P. Sprechmann, K. Perlin, Accelerating eulerian fluid simulation with convolutional networks, in: International Conference on Machine Learning, PMLR, 2017, pp. 3424–3433.

[28] E. Ajuria Illarramendi, A. Alguacil, M. Bauerheim, A. Misdariis, B. Cuenot, E. Benazera, Towards an hybrid computational strategy based on deep learning for incompressible flows, in: AIAA Aviation 2020 Forum, 2020, p. 3058.

[29] R. Chen, X. Jin, H. Li, A machine learning based solver for pressure poisson equations, Theoretical and Applied Mechanics Letters 12 (5) (2022).

[30] A. J. Chorin, The numerical solution of the navier-stokes equations for an incompressible fluid, Bulletin of the American Mathematical Society 73 (6) (1967) 928–931.

[31] J. B. Perot, An analysis of the fractional step method, Journal of Computational Physics 108 (1) (1993) 51–58.

[32] A. Kaneda, O. Akar, J. Chen, V. A. T. Kala, D. Hyde, J. Teran, A deep conjugate direction method for iteratively solving linear systems, in: International Conference on Machine Learning, PMLR, 2023, pp. 15720–15736.

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

[34] W. L. Briggs, V. E. Henson, S. F. McCormick, A multigrid tutorial, 2nd Edition, SIAM Press, 2015.

[35] G. H. Golub, C. F. V. Loan, Matrix computation, 4th Edition, The Johns Hopkins University Press, Baltimore, 2013.

[36] E. Zhang, A. Kahana, A. Kopaničáková, E. Turkel, R. Ranade, J. Pathak, G. E. Karniadakis, Blending neural operators and relaxation methods in PDE numerical solvers, Nature Machine Intelligence (2024) 1–11.

[37] A. Kahana, E. Zhang, S. Goswami, G. Karniadakis, R. Ranade, J. Pathak, On the geometry transferability of the hybrid iterative numerical solver for differential equations, Computational Mechanics 72 (3) (2023) 471–484.

[38] W. Suo, W. Zhang, A novel paradigm for solving pdes: multi-scale neural computing, Acta Mechanica Sinica 41 (6) (2025) 324172.

[39] A. Kopaničáková, G. E. Karniadakis, Deeponet based preconditioning strategies for solving parametric linear systems of equations, SIAM Journal on Scientific Computing 47 (1) (2025) C151–C181.

[40] D. Dong, W. Suo, J. Kou, W. Zhang, PINN-MG: A multigrid-inspired hybrid framework combining iterative method and physics-informed neural networks, arXiv preprint arXiv:2410.05744 (2024).

[41] R. Chen, X. Jin, N. A. Adams, H. Li, High-efficient machine learning projection method for incompressible navier-stokes equations, arXiv preprint arXiv:2501.13966 (2025).

[42] J. Nocedal, S. J. Wright, Numerical optimization, 2nd Edition, Springer, 2006.

[43] J. R. Shewchuk, et al., An introduction to the conjugate gradient method without the agonizing pain (1994).

[44] R.-Y. Li, C.-M. Xie, W.-X. Huang, C.-X. Xu, An efficient immersed boundary projection method for flow over complex/moving boundaries, Computers & Fluids 140 (2016) 122–135.

[45] J. K. Dukowicz, A. S. Dvinsky, Approximate factorization as a high order splitting for the implicit incompressible flow equations, Journal of Computational Physics 102 (2) (1992) 336–347.

[46] C. S. Peskin, Flow patterns around heart valves: a numerical method, Journal of Computational Physics 10 (2) (1972) 252–271.

[47] C. T. Taira K, The immersed boundary method: a projection approach, Journal of Computational Physics 225 (2) (2007) 2118–2137.

[48] A. M. Roma, C. S. Peskin, M. J. Berger, An adaptive version of the immersed boundary method, Journal of computational physics 153 (2) (1999) 509–534.

[49] P.-Y. Chuang, O. Mesnard, A. Krishnan, L. A Barba, Petibm: toolbox and applications of the immersed-boundary method on distributed-memory architectures, Journal of Open Source Software 3 (25) (2018).

[50] C. Lanczos, An iteration method for the solution of the eigenvalue problem of linear differential and integral operators, Journal of research of the National Bureau of Standards 45 (4) (1950) 255–282.

[51] O. Ronneberger, P. Fischer, T. Brox, U-net: Convolutional networks for biomedical image segmentation, in: Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18, Springer, 2015, pp. 234–241.

[52] K. W. Lan, E. Gueidon, A. Kaneda, J. Panetta, J. Teran, A neural-preconditioned poisson solver for mixed dirichlet and neumann boundary conditions, arXiv preprint arXiv:2310.00177 (2023).

[53] N. Wandel, M. Weinmann, M. Neidlin, R. Klein, Spline-pinn: Approaching pdes without data using fast, physics-informed hermite-spline cnns, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 36, 2022, pp. 8529–8538.

[54] P. Foret, A. Kleiner, H. Mobahi, B. Neyshabur, Sharpness-aware minimization for efficiently improving generalization, arXiv preprint arXiv:2010.01412 (2020).

[55] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. Gropp, et al., Petsc users manual (2019).

[56] D. M. W. Jakob, J. Rhinelander, pybind11 — seamless operability between c++11 and python, `https://github.com/pybind/pybind11` (2017).

[57] U. Ghia, K. N. Ghia, C. Shin, High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method, Journal of Computational Physics 48 (3) (1982) 387–411.

[58] Y. Saad, Iterative methods for sparse linear systems, SIAM, 2003.