Computers and Fluids 316 (2026) 107154

Contents lists available at ScienceDirect

Computers and Fluids

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

Surrogate modeling of fluid flow under different conditions using physics-informed Deep Operator Networks☆

Junya Onishi [a,*], Harutaka Kitagawa [b], Rishabh Puri [c,e], Mario Rüttgers [d,e], Rakesh Sarma [e], Andreas Lintermann [e], Makoto Tsubokura [a,b]

[a] RIKEN Center for Computational Science, 7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, 650-0047, Hyogo, Japan
[b] Graduate School of System Informatics, Kobe University, 1-1 Rokkodai-cho, Nada-ku, Kobe, 657-8501, Hyogo, Japan
[c] Simulation of Reacting Thermo-Fluid Systems, Engler-Bunte-Institute, Karlsruhe Institute of Technology, Engler-Bunte Ring 7, Karlsruhe, 76131, Germany
[d] Data-Driven Fluid Engineering Laboratory, Inha University, 100, Inha-ro, Michuhol-gu, Incheon, 22212, Republic of Korea
[e] Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, Wilhelm-Johnen-Straße, Jülich, 52425, Germany

ARTICLE INFO

Keywords:
Surrogate modeling
Physics-informed neural networks (PINNs)
Deep Operator Network (deepONet)

ABSTRACT

We applied and evaluated a physics-informed Deep Operator Network (PI-DeepONet) for modeling incompressible two-dimensional steady flows under varying Reynolds numbers and inlet boundary conditions. By combining the generalization capability of DeepONets with the physical constraints imposed by physics-informed neural networks (PINNs), the framework enables flow field prediction without relying on labeled data. Two types of input variations are considered: parametric variation in Reynolds numbers and functional variation in inlet velocity profiles. The results show that PI-DeepONet successfully generalizes across both scenarios, accurately predicting velocity and pressure fields even for unseen configurations. Furthermore, we explored the impact of architectural design on performance and found that shared-network variants significantly reduce computational cost without sacrificing accuracy. These results highlight both the potential and limitations of PI-DeepONet as a practical surrogate modeling tool for scientific computing.

1. Introduction

Computer-aided engineering (CAE) is an essential tool for modern product development, providing comprehensive support throughout the entire design cycle. As the demand for high-fidelity modeling and rapid design iterations continues to grow, the computational cost and complexity of simulations have increased accordingly. Moreover, CAE is no longer used solely as a substitute for physical experiments. It is now widely applied to more advanced tasks such as design optimization, real-time control, and uncertainty quantification. These emerging applications impose new demands on simulation frameworks, which must deliver high accuracy, computational efficiency, and strong generalization capabilities across diverse conditions. In particular, problems governed by complex partial differential equations, such as those encountered in computational fluid dynamics (CFD), remain a significant challenge.

Several strategies have traditionally been pursued to address the computational burden in CFD simulations. One common approach is reduced-order modeling (ROM). Classical ROM techniques, such as Proper Orthogonal Decomposition (POD) and Galerkin projection, reduce the dimensionality of the governing equations by projecting the full-order solution onto a set of low-dimensional modes derived from high-fidelity snapshots [1,2]. A typical example of such parametric reduced-order modeling can be found in the work of Rouizi et al. [3], who developed a reduced-order model for the two-dimensional steady backward-facing step flow by treating the Reynolds number as a varying parameter. While these methods are computationally efficient and analytically interpretable, they often suffer from poor generalization when boundary conditions or physical parameters vary, and typically require intrusive reformulations of the underlying PDEs.

To overcome these limitations, data-driven ROM approaches, leveraging machine learning (ML) techniques, have attracted increasing attention. Methods based on regression, kernel techniques, and deep learning architectures, such as autoencoders and latent-space models, have been proposed to learn low-dimensional representations directly from simulation data [4]. Although these methods are non-intrusive and flexible, they generally require large datasets and tend to significantly degrade when extrapolating beyond the training distribution.

☆ This article is part of a Special issue entitled: 'fusing data and physics' published in Computers and Fluids.

* Corresponding author.
E-mail address: junya.onishi@riken.jp (J. Onishi).

https://doi.org/10.1016/j.compfluid.2026.107154
Received 30 April 2025; Received in revised form 20 February 2026; Accepted 22 May 2026
Available online 28 May 2026

Recently, the field of physics-informed machine learning (PIML) has emerged as a promising framework that integrates physical principles directly into machine learning models [5,6]. PIML aims to improve the robustness and generalization of data-driven models by embedding physical constraints into model training, such as enforcing conservation laws, symmetry properties, or governing equation residuals. A prominent and widely studied example is physics-informed neural networks (PINNs) [7], which offer a powerful framework that embeds the governing equations of physical systems directly into the loss function. This hybrid formulation allows the network to learn solutions consistent with physical laws, even when only sparse or incomplete data are available. Importantly, PINNs can be flexibly applied to both forward and inverse problems, enabling the discovery of unknown parameters, boundary conditions, or hidden dynamics in addition to solving forward simulations. They have been successfully applied to various types of fluid flows, including laminar flows [7–9], turbulent flows [10], high-speed flows [11], biomedical flows [12], and multiphase flows [13] (see the review [14] and references therein). However, their generalization capability is limited, as they are typically trained for specific configurations and require retraining whenever boundary conditions or parameters change.

To improve the generalization capability, recent studies have turned toward operator learning, which seeks to approximate mappings between function spaces rather than between finite-dimensional vectors. Among such models, the Deep Operator Network (DeepONet) [15] was proposed to learn nonlinear solution operators from input functions, such as initial or boundary conditions, to output functions governed by PDEs. A key feature of DeepONet is its two-network structure: a branch network processes the input function, while a trunk network encodes the spatial (or temporal) coordinates where the solution is evaluated. The final output is computed as an inner product between the outputs of these two networks. This separation allows the model to flexibly incorporate both functional and spatial information, and facilitates generalization across different input conditions. Another notable approach is the Fourier Neural Operator (FNO) [16], which achieves high accuracy and computational efficiency by applying convolutions in the Fourier domain. FNO learns solution operators by transforming the input functions to spectral space, applying learned linear transformations, and transforming them back via inverse Fourier transform. This spectral formulation enables the model to capture long-range dependencies with fewer parameters compared to traditional CNN-based architectures, and has been successfully applied to various PDEs. While these models exhibit strong generalization across inputs, they do not explicitly enforce physical constraints, which can lead to inconsistent or non-physical predictions in extrapolative regimes.

Building on this perspective, the present study investigates a hybrid modeling approach that integrates physical constraints into the DeepONet architecture, resulting in what is known as physics-informed Deep Operator Networks (PI-DeepONets), as originally proposed by Wang et al. [17]. This framework seeks to leverage the generalization capability of operator learning while ensuring the physical consistency through the explicit incorporation of governing equations during training. Their results demonstrated that this approach enables accurate and physically consistent operator learning, even in the absence of labeled training data. To further improve the robustness and predictive performance of PI-DeepONets, Wang et al. [18] proposed a series of architectural and algorithmic improvements. These include adaptive loss reweighting to dynamically balance different loss components, and gradient stabilization techniques to mitigate training instabilities. They also showed that these modifications lead to more accurate and efficient learning, particularly for nonlinear PDEs with complex solution landscapes. To address the scalability limitations of operator learning in high-dimensional settings, Mandl et al. [19] introduced Separable DeepONets. This approach reformulates the trunk-branch interaction using a separable representation, allowing the model to decompose complex operators into lower-dimensional components. By reducing the number of trainable parameters and computational cost, while preserving expressive capacity, Separable DeepONets offer a promising direction for extending physics-informed operator learning to high-dimensional and multi-scale problems.

Recent developments have also extended the physics-informed operator learning paradigm to broader neural architectures, such as FNO [20], opening new avenues for combining spectral representations with physical constraints. In parallel, alternative formulations specifically tailored to gradient-flow systems have been proposed. Zhang et al. [21] introduced an energy-dissipative evolutionary DeepONet (EDE-DeepONet), which enforces energy dissipation by modeling the temporal evolution of network parameters through auxiliary ODEs. Similarly, Li et al. [22] developed Phase-Field DeepONet, a framework that incorporates the minimizing movement scheme into DeepONet training to approximate the time evolution of pattern-forming systems governed by gradient flows. While these approaches are highly efficient for energy-dissipative systems, they avoid explicit residual minimization and are inherently limited to problems that can be formulated as gradient flows of free-energy functionals.

As mentioned above, research on PI-DeepONets has primarily focused on methodological advances, while application-oriented studies remain relatively limited. In particular, the application of PI-DeepONets to realistic fluid flow problems is still scarce, especially in scenarios involving systematic variations in physical parameters or boundary conditions. Such variations are ubiquitous in practical settings and pose significant challenges for surrogate models in terms of generalization and robustness. To address this gap, we systematically evaluate the generalization capability of the PI-DeepONet framework in the context of steady incompressible flows. We consider two representative types of input variation: changes in Reynolds number (scalar input) and changes in inlet velocity profile (functional input). These cases are selected to reflect simple but challenging conditions in fluid dynamics and to examine the model's ability to generalize across both parametric and functional input spaces. Through controlled comparisons with conventional PINNs, this study highlights the strengths and limitations of physics-informed operator learning and provides a foundation for developing unified, efficient, and physically consistent surrogate models for diverse flow conditions.

The remainder of this paper is organized as follows. Section 2 introduces the overall modeling framework and describes the network architecture, training strategy, and input encoding for both Reynolds number variation and inlet boundary condition variation. Section 3 presents numerical results for each case, including accuracy evaluations, generalization tests, and comparisons among different network architectures. Finally, Section 4 concludes the paper with a summary and discussion.

2. Method

The primary objective of this study is to evaluate the performance of PI-DeepONets as a surrogate modeling framework for fluid flows under varying conditions. To this end, we consider two distinct problem settings as follows.

- **Case 1: Backward-facing step flow with varying Reynolds numbers.** In this problem, the Reynolds number Re is treated as a scalar input to the model. While the DeepONet framework is originally designed to learn mappings from function spaces to function spaces, we deliberately adopt the PI-DeepONet architecture here for two reasons: (i) to maintain architectural consistency with Case 2, which involves functional inputs, and (ii) to investigate whether the separation of spatial and parametric representations, an essential feature of DeepONets, offers advantages even when the input is a scalar. It should be noted that conventional feed-forward neural networks could also handle this setting; the use of PI-DeepONet is not essential but intentional for comparative purposes.

- **Case 2: Channel flow with varying inlet velocity profiles.** In contrast to Case 1, this problem involves function-valued inputs: the inlet velocity profile is provided in discretized form along the inflow boundary. This setting enables the assessment of the model's generalization capability across functional boundary conditions.

These two cases serve as representative examples for evaluating the ability of PI-DeepONet to generalize across both scalar and functional input variations. Details of the formulation, network architecture, and training procedures for each case are described in the subsequent sections.

2.1. Modeling two-dimensional steady-state flow using PINNs

To formulate the PINN-based approach, we begin with the two-dimensional steady-state incompressible Navier–Stokes equations:

$$\nabla \cdot \boldsymbol{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \tag{1}$$

$$\boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \boldsymbol{u}, \tag{2}$$

where $\boldsymbol{u} = (u, v)$ is the velocity field, p is the pressure, ρ is the mass density, and ν is the kinematic viscosity. For the present study, the mass density is assumed constant and set to unity without loss of generality. These equations are enforced throughout the domain to ensure physical consistency of the learned solution.

To approximate the solution fields (u, v, p), we construct a fully connected feedforward neural network (FNN) that takes the spatial coordinates $\boldsymbol{x} = (x, y)$ as input. The network outputs a vector $\boldsymbol{y} = (u_\theta, v_\theta, p_\theta)$, which represents the predicted velocity and pressure fields. Here, θ denotes the trainable parameters of the network.

The network consists of N_L layers, each performing an affine transformation followed by a nonlinear activation, except for the final output layer. Each layer has N_W neurons. The forward pass is written as:

$$\boldsymbol{h}^{(0)} = \boldsymbol{x}, \tag{3}$$

$$\boldsymbol{h}^{(l)} = \sigma\left(\boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}\right), \quad l = 1, \ldots, N_L - 1, \tag{4}$$

$$\boldsymbol{y} = \boldsymbol{W}^{(N_L)}\boldsymbol{h}^{(N_L-1)} + \boldsymbol{b}^{(N_L)}, \tag{5}$$

where $\boldsymbol{h}^{(l)}$ is the lth hidden layer, and σ is the activation function. The quantities $\boldsymbol{W}^{(l)}$ and $\boldsymbol{b}^{(l)}$ represent the weight matrix and bias vector respectively, which are the trainable parameters at the lth layer.

In this study, the hyperbolic tangent function $\tanh(\cdot)$ is used as the activation function throughout the network. This choice is motivated by its smoothness and differentiability, which are beneficial when computing derivatives via automatic differentiation [23]. Other activation functions may also be considered, depending on the problem setting and numerical requirements.

Automatic differentiation is employed to compute the derivatives of the output with respect to the input, enabling the evaluation of the spatial derivatives of the flow variables. The results are then substituted into the governing equations to evaluate the residuals. The physics loss is defined as the mean squared residual of the equations over N_f collocation points $\{\boldsymbol{x}_f^{(i)}\}_{i=1}^{N_f}$:

$$\mathcal{L}_{\mathrm{PDE}} = \frac{1}{N_f}\sum_{i=1}^{N_f}\left(r_{\mathrm{cont}}^2(\boldsymbol{x}_f^{(i)}) + r_u^2(\boldsymbol{x}_f^{(i)}) + r_v^2(\boldsymbol{x}_f^{(i)})\right), \tag{6}$$

where $r_{\mathrm{cont}}, r_u, r_v$ represent the residuals of the continuity and momentum equations, Eqs. (1) and (2), respectively.

In addition, the boundary loss $\mathcal{L}_{\mathrm{BC}}$ is defined as the sum of mean squared errors (MSEs) evaluated at boundary points $\{\boldsymbol{x}_b^{(i)}\}_{i=1}^{N_b}$, enforcing the corresponding Dirichlet or Neumann conditions for velocity and pressure at each boundary. These conditions are incorporated into the loss function as follows:

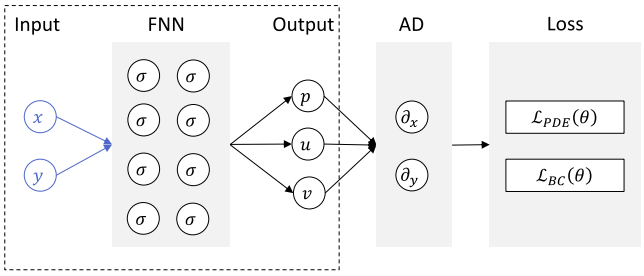

Fig. 1. Architecture of the physics-informed Neural Network (PINN) designed for two-dimensional steady-state flow problems. The network takes spatial coordinates (x, y) as input and outputs the velocity components (u, v) and pressure p. Automatic differentiation is used to compute the derivatives of the outputs with respect to the inputs, which are then used to evaluate the residuals of the governing equations. In this study, the total loss consists of the equation residual loss and the boundary condition loss, and the network is trained to minimize this combined loss.

$$\mathcal{L}_{\mathrm{BC}} = \mathcal{L}_{\mathrm{inlet}} + \mathcal{L}_{\mathrm{outlet}} + \mathcal{L}_{\mathrm{wall}}, \tag{7}$$

where $\mathcal{L}_{\mathrm{inlet}}$, $\mathcal{L}_{\mathrm{outlet}}$, and $\mathcal{L}_{\mathrm{wall}}$ are the loss terms corresponding to the inlet, outlet, and wall boundaries, respectively. In the present study, the inlet loss enforces a prescribed velocity profile for the velocity components and a homogeneous Neumann condition for pressure. The outlet loss enforces a Dirichlet condition for pressure and zero gradients for the velocity components. The wall loss imposes a no-slip condition on the velocity and a zero normal derivative condition for pressure. The resulting loss terms are given as follows:

$$\mathcal{L}_{\mathrm{inlet}} = \frac{1}{N_b^{\mathrm{inlet}}}\sum_{i=1}^{N_b^{\mathrm{inlet}}}\left[\left(u_\theta(\boldsymbol{x}_b^{(i)}) - u_{\mathrm{in}}(\boldsymbol{x}_b^{(i)})\right)^2 + \left(v_\theta(\boldsymbol{x}_b^{(i)})\right)^2 + \left(\frac{\partial p_\theta}{\partial n}(\boldsymbol{x}_b^{(i)})\right)^2\right], \tag{8}$$

$$\mathcal{L}_{\mathrm{outlet}} = \frac{1}{N_b^{\mathrm{outlet}}}\sum_{i=1}^{N_b^{\mathrm{outlet}}}\left[\left(\frac{\partial u_\theta}{\partial n}(\boldsymbol{x}_b^{(i)})\right)^2 + \left(\frac{\partial v_\theta}{\partial n}(\boldsymbol{x}_b^{(i)})\right)^2 + \left(p_\theta(\boldsymbol{x}_b^{(i)})\right)^2\right], \tag{9}$$

$$\mathcal{L}_{\mathrm{wall}} = \frac{1}{N_b^{\mathrm{wall}}}\sum_{i=1}^{N_b^{\mathrm{wall}}}\left[\left(u_\theta(\boldsymbol{x}_b^{(i)})\right)^2 + \left(v_\theta(\boldsymbol{x}_b^{(i)})\right)^2 + \left(\frac{\partial p_\theta}{\partial n}(\boldsymbol{x}_b^{(i)})\right)^2\right], \tag{10}$$

where the values $N_b^{\mathrm{inlet}}, N_b^{\mathrm{outlet}}, N_b^{\mathrm{wall}}$ represent the number of boundary collocation points on the inlet, outlet, and wall, respectively. The term $\partial/\partial n$ indicates differentiation in the direction normal to the boundary.

The total loss is defined as:

$$\mathcal{L}_{\mathrm{total}} = \mathcal{L}_{\mathrm{PDE}} + \mathcal{L}_{\mathrm{BC}}. \tag{11}$$

All the models are trained by minimizing the total loss function defined in Eq. (11). The number and spatial distribution of collocation points, both in the interior (N_f) and on the boundaries ($N_b^{\mathrm{inlet}}, N_b^{\mathrm{wall}}, N_b^{\mathrm{outlet}}$), depend on the specific problem configuration and will be described in later sections.

In general, balancing multiple loss terms through weighting coefficients can be important in physics-informed learning, particularly when the magnitudes of the individual components differ significantly [18]. In the present study, however, no additional weighting coefficients were introduced. In other words, both $\mathcal{L}_{\mathrm{PDE}}$ and $\mathcal{L}_{\mathrm{BC}}$ were assigned equal weights. A quantitative assessment of the relative contributions of the PDE and boundary losses is provided in Section 3.1.2.

The overall architecture and training process of this PINN formulation are illustrated in Fig. 1. To enable generalization with respect to varying conditions, we extend the network architecture to

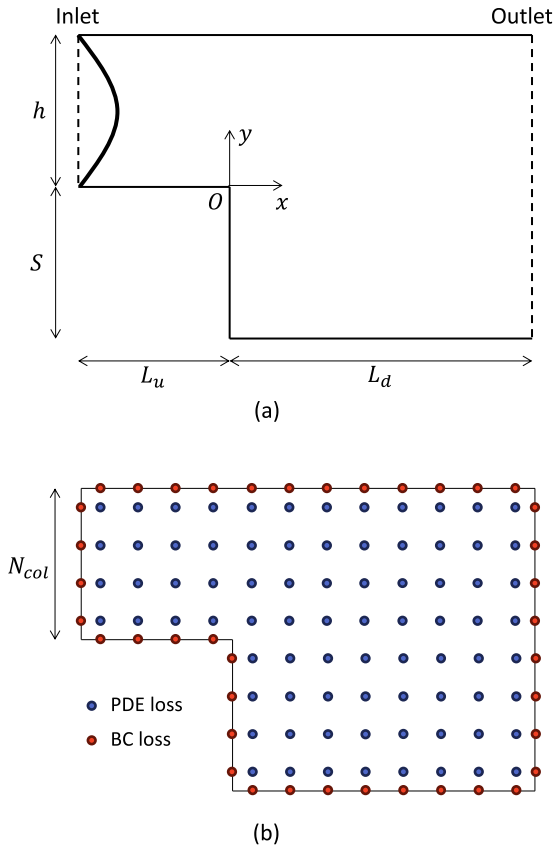

Fig. 2. (a) Schematic illustration of the computational domain and loss function regions. The domain consists of a rectangular channel with an inlet, outlet, and solid walls. (b) Distribution of the collocation points used for loss evaluation. The PDE loss is evaluated in the interior of the domain, while the boundary condition loss is imposed along the domain boundaries. For visualization purposes, only a representative subset of the collocation points is displayed.

a physics-informed Deep Operator Network (PI-DeepONet) in the following sections. Importantly, this extension modifies only the neural network component shown in Fig. 1, while the underlying PINN formulation, including the governing equations and loss computation, remains unchanged.

2.2. Case 1: Generalization across Reynolds numbers

As described in Section 2.1, the baseline formulation employs a PINN to approximate the steady-state solution of two-dimensional incompressible Navier–Stokes equations. To enable generalization across varying Reynolds numbers without retraining, we extend this formulation to a PI-DeepONet. In this section, we present the problem setting, network architecture, and training configuration used for Case 1, where the Reynolds number is treated as an input parameter. The goal is to construct a physics-informed surrogate model that enables accurate prediction of velocity and pressure fields across a range of Reynolds numbers, without requiring retraining.

2.2.1. Problem description

Case 1 considers a two-dimensional backward-facing step flow under steady-state and incompressible conditions. The computational domain, illustrated in Fig. 2(a), consists of a horizontal channel with a sudden expansion at the bottom wall. The inlet section has height h, while the outlet section expands to height $h + S$. The expansion ratio of the channel height is given by $(h+S)/h = 1.9423$. The upstream and downstream lengths are denoted by L_u and L_d, respectively. In all configurations, the upstream length is fixed at $L_u/h = 1$. These dimensions follow previous studies on backward-facing step flow and are chosen to ensure sufficient development of the recirculation region [24].

The flow enters from the left boundary, where a parabolic velocity profile is imposed in the streamwise direction:

$$u_{\text{in}}(y) = 6U_b \frac{y}{h}\left(1 - \frac{y}{h}\right),$$

where U_b is the bulk velocity of the inlet flow. Based on this bulk velocity, the Reynolds number is defined as

$$\text{Re} = \frac{U_b D}{\nu},$$

where $D = 2h$ is the hydraulic diameter of the inlet channel, and ν is the kinematic viscosity of the fluid.

2.2.2. Network architecture

In this setting, the Reynolds number Re is treated as a scalar input to the model, while the spatial coordinates (x, y) serve as evaluation points for the solution fields. This is achieved by embedding Re as an input and comparing several network architectures with varying levels of modularity and inductive bias. To incorporate the Reynolds number as an input parameter, we compare the following architectures: a standard PINN, a PINN with decoupled outputs, and a PI-DeepONet model. The structure of each architecture is illustrated in Fig. 3:

- **PINN (baseline):** The Reynolds number Re is appended directly to the spatial input, forming a three-dimensional input vector (x, y, Re). A single fully connected neural network with N_L layers and N_W neurons per layer maps this input to the output variables (u, v, p). While simple to implement, this approach does not distinguish the differing physical characteristics of each output variable.
- **Split-output PINN:** An independent neural network is assigned to each output variable, u, v, and p, with all networks receiving the same input (x, y, Re). The networks have the same architectural parameters defined by N_L layers and N_W neurons per layer. By isolating the outputs, each network can specialize in capturing the features relevant to its target variable. This approach serves as a bridge between the standard PINN and the operator-based formulation introduced below.
- **PI-DeepONet:** The PI-DeepONet architecture employs a DeepONet [15] structure in which the spatial coordinates (x, y) are fed into a trunk network, and the Reynolds number Re is provided to a branch network. The final output is given by the inner product of the outputs of the trunk and branch networks. Since DeepONet yields a single output per trunk-branch pair, separate pairs are constructed for each of the output variables u, v, and p. Both the trunk and branch networks are implemented as fully connected neural networks with identical architecture, consisting of N_L layers and N_W neurons per layer. This formulation allows the model to decouple spatial representation from parametric dependence, potentially enhancing generalization.
  Recent extensions of DeepONet have proposed alternative formulations for vector-valued field prediction. For instance, tensor-product variants replace the standard inner-product operation with higher-order feature interactions, enabling multi-component outputs within a single trunk-branch pair [25]. Sequential DeepONet architectures have also been introduced for transient problems, where operator learning is combined with autoregressive time integration [26]. In contrast, the present study focuses on steady-state flow problems and adopts a standard inner-product-based formulation, constructing independent trunk-branch pairs for each physical variable. This design prioritizes architectural simplicity and parameter efficiency while retaining sufficient expressiveness for steady vector-field prediction.

In addition, we explore two variants of PI-DeepONet, as shown in Fig. 4. Both configurations aim to reduce the model complexity in terms of the number of trainable parameters and memory footprint, while maintaining the capacity of the original PI-DeepONet to learn distinct mappings for each physical quantity.

- **PI-DeepONet, Branch-sharing configuration:** A single branch network processes the Reynolds number Re, and its output is shared across three separate trunk networks corresponding to u, v, and p.
- **PI-DeepONet, Trunk-sharing configuration:** A single trunk network processes the spatial coordinates (x, y), and its output is shared across three separate branch networks corresponding to u, v, and p.

In all architectures, each FNN is configured with the same number of layers N_L and neurons per layer N_W. This also applies to the PI-DeepONet, where both the trunk and branch networks, denoted FNN(T) and FNN(B) respectively, share the same configuration. Performance differences across architectures are discussed in Section 3.1.2.

2.2.3. Training configuration

All collocation points are sampled uniformly in the interior of the domain for the PDE residual loss, and along the remaining boundaries (walls and outlet) for the respective boundary loss terms, as shown in Fig. 2(b). Specifically, a total of $N_x \times N_y = 200 \times 100$ collocation points are initially placed on a uniform grid, corresponding to cell centers in the finite volume sense. Subsequently, the collocation points located below the backward-facing step are excluded from the domain. As a result, the number of interior collocation points used for training is $N_f = 17,550$. In addition, $N_b^{\text{inlet}} = 51$ points are placed along the inlet boundary to enforce Dirichlet conditions on the velocity components, and $N_b^{\text{outlet}} = 100$ points are used for the outlet boundary condition. For the wall boundaries, a total of $N_b^{\text{wall}} = 200 + 200 + 49$ points are used. These point densities, both in the interior and at the inlet, were chosen with reference to the resolution employed in Biswas et al. [24], which provided sufficient accuracy in conventional multigrid simulations of backward-facing step flows. Although this choice does not guarantee similar performance in the PINN framework, it serves as a practical starting point under comparable geometric conditions. It should also be noted that PINNs involve other sources of error, such as training variance and optimization-related effects, which may influence accuracy independently of point density.

All networks are trained using the same set of collocation and boundary points to ensure consistent comparison across architectures.

To assess the model's ability to generalize with respect to the Reynolds number, we consider two training scenarios.

- **Case A:** 11 Reynolds numbers are sampled uniformly from Re = 1 to 100. The downstream length is set to $L_d/h = 3$ in this case.
- **Case B:** 11 Reynolds numbers are sampled uniformly from Re = 100 to 400. A longer downstream length of $L_d/h = 9$ is used to accommodate the extended recirculation zone expected at higher Reynolds numbers.

These two configurations are selected to investigate the effects of the range of Reynolds numbers used in the training, and to evaluate the model's ability to generalize within and beyond the training range. It should be noted that the kinematic viscosity values ν are calculated to match the specified Reynolds numbers, $\text{Re} = U_b D/\nu$, while the characteristic velocity U_b and reference length D are fixed. These values of ν are then used to compute the residual loss $\mathcal{L}_{\text{PDE}}$ in the training of the networks. It should also be noted that even at training Reynolds numbers (e.g., Re = 1), no reference solutions are provided as training data. The training is conducted in a physics-informed manner, based solely on the governing equations.

Each FNN, as described in Section 2.2.2, consists of $N_L = 4$ hidden layers, each with $N_W = 64$ neurons.

All models are implemented in Python using the PyTorch framework. Optimization is performed using the Adam optimizer with a fixed learning rate of 1×10^{-3}. The network weights are initialized using Xavier initialization. Training is conducted for a fixed number of 500,000 epochs, which was determined empirically and held constant across all models. Full-batch training is employed, meaning that all collocation and boundary points are used in every optimization step.

To perform the above training, a single NVIDIA A100 GPU (40 GB), on the FUJITSU Server PRIMERGY GX2570 (Wisteria/BDEC-01) at the Information Technology Center, The University of Tokyo, was used.

2.3. Case 2: Generalization across inlet conditions

2.3.1. Problem description

To investigate the generalization capability with respect to boundary conditions, we consider a two-dimensional incompressible channel flow bounded by fixed walls at the top and bottom boundaries. The computational domain is illustrated in Fig. 5. At the inlet boundary, an arbitrary velocity profile is prescribed for the streamwise velocity component, while the wall-normal component is set to zero.

Under these conditions, the objective is to evaluate whether the PI-DeepONet can accurately predict the resulting steady-state flow field for a given inlet velocity distribution.

2.3.2. Network architecture

To enable generalization with respect to boundary conditions, we extend the PI-DeepONet framework to handle varying inlet velocity profiles as inputs. Unlike the case where the Reynolds number is used as a scalar parameter, the boundary condition here is a function defined along the inlet boundary. This functional input is incorporated into the model through the branch network of the DeepONet architecture.

Fig. 6 illustrates the PI-DeepONet design for modeling flow behavior under varying inlet conditions. The branch network takes as input a discretized inlet velocity profile, sampled at a finite number of sensor points along the inflow boundary, while the trunk network receives the spatial coordinates (x, y) where predictions are to be made. The final output is obtained as the inner product of the trunk and branch network outputs, yielding predictions for the velocity components (u, v), and pressure p.

During training, as described above, the discretized inlet velocity profile is provided to the branch network as a functional input. This input profile is then compared with the predicted velocity at the inlet, and the resulting mean squared error is incorporated into the boundary loss, together with other boundary terms, as defined in Eq. (11). In this manner, PI-DeepONet learns to adapt to varying boundary conditions provided as functional inputs. By contrast, in conventional PINNs, the network is trained to satisfy a fixed set of boundary conditions specified a priori.

In Case 2, we evaluate only the base PI-DeepONet architecture. This is because the primary objective here is to assess the generalization capability of operator learning under function-valued boundary conditions, rather than to benchmark multiple network designs. The architectural variants of PI-DeepONet, such as trunk-sharing and branch-sharing, are expected to be beneficial, especially in reducing memory usage associated with high-dimensional functional inputs. However, their implementation is beyond the scope of this study and is left for future work.

2.3.3. Training configuration

The model is trained in a fully physics-informed manner, without supervised flow data. A total of $N_x \times N_y = 50 \times 50 (= N_f)$ collocation points are uniformly distributed within the interior of the domain. In addition, $N_b^{\text{inlet}} = N_b^{\text{outlet}} = 50$ collocation points are placed uniformly

Table 1
Definition and summary statistics of features computed from 800 RBF-based inlet velocity profiles.

Feature	Definition	Mean	Std. Dev.	Min	Max
Maximum value	$\max_y u_{\text{in}}(y)$	1.48	1.07	0.00	5.66
Minimum value	$\min_y u_{\text{in}}(y)$	−1.63	1.08	−6.48	0.00
Range	max − min	3.11	1.16	0.77	7.86
Mean value	$\frac{1}{H}\int_0^H u_{\text{in}}(y)\,dy$	−0.06	0.72	−2.12	2.01
Energy	$\int_0^H u_{\text{in}}^2(y)\,dy$	1.56	1.32	0.04	10.11
RMS	$\sqrt{\frac{1}{H}\int_0^H u_{\text{in}}^2(y)\,dy}$	0.93	0.38	0.21	2.53
Skewness	$\frac{1}{H\sigma^3}\int_0^H (u_{\text{in}}(y)-\mu)^3\,dy$	0.00	1.06	−4.00	4.91

along each of the inlet and outlet boundaries, and a total of $N_b^{\text{wall}} = 50+50$ collocation points are used along the top and bottom boundaries.

To generate diverse inlet velocity profiles, we construct them as linear combinations of N_{basis} Gaussian radial basis functions (RBFs) [15], each centered at one of the uniformly distributed sensor locations $y^{(i)}$ along the inlet boundary. Let $\hat{y} = y/H$ denote the normalized wall-normal coordinate. Each basis function is then defined as

$$f_i(\hat{y}) = \exp\left(-\frac{(\hat{y}-\hat{y}^{(i)})^2}{2l^2}\right), \quad i = 1, \ldots, N_{\text{basis}},$$

where l is a free parameter that controls the length scale of the generated profiles. The inlet velocity profile is formed as

$$u_{\text{in}}(y) = 4U_0\left(\sum_{i=1}^{N_{\text{basis}}} w_i f_i(\hat{y})\right)\cdot \hat{y}(1-\hat{y}),$$

where U_0 is a velocity scale, and $\mathbf{w} = (w_1, \ldots, w_{N_{\text{basis}}})$ is a weight vector sampled uniformly from the interval $[-1, 1]$. The multiplicative term $\hat{y}(1-\hat{y})$ ensures that the no-slip boundary condition is satisfied at the top and bottom walls, i.e., $u_{\text{in}}(0) = u_{\text{in}}(H) = 0$. In this study, the length scale of the Gaussian kernel is fixed at $l/H = 0.1$, providing a balance between localization and smoothness in the generated profiles. The velocity scale is fixed at $U_0 = 1$ in all training samples.

We set $N_{\text{basis}} = 50$ to match the number and spatial distribution of sensor locations along the inlet boundary. A total of $N_{\text{profiles}} = 800$ weight vectors are generated during training, resulting in the same number of distinct inlet velocity profiles. The value of N_{profiles} is determined by the maximum capacity supported by the GPU memory. To facilitate efficient training with such a large number of functional inputs, we employ 8 GPUs in parallel using the Distributed Data Parallel (DDP) module implemented in PyTorch.

For each inlet profile, the velocity distribution is sampled at the predefined sensor locations along the inlet boundary. These sampled values constitute a finite-dimensional representation of the inlet function and are provided as inputs to the branch network. Thus, the branch network receives a discretized inlet velocity profile rather than the RBF coefficients directly, which is consistent with the standard DeepONet formulation for operator learning.

Table 1 summarizes the statistical characteristics of the training inlet profiles. Constructed as random linear combinations of RBFs, these profiles exhibit a wide range of shapes and amplitudes. The table provides a quantitative overview of their magnitude, energy, symmetry, and variability.

The remaining training details are the same as those used in Case 1. Each FNN consists of $N_L = 4$ hidden layers with $N_W = 64$ neurons per layer and tanh activation functions. All weights are initialized using Xavier initialization. Training is performed using the full-batch Adam optimizer for 100,000 epochs with a learning rate of 10^{-3}. No minibatching is applied; the entire training set is used in each iteration.

Table 2
Comparison of training time, memory usage, and number of FNNs for different architectures. Values in parentheses indicate ratios relative to the baseline PINN. Memory usage was measured using `PyTorch MemLab`.

Model	Time [s]	Memory	# FNNs
PINN	725	149 MB	1
Split-output PINN	1968 (2.7)	438 MB (2.9)	3
PI-DeepONet	2569 (3.5)	1.12 GB (7.6)	6
PI-DeepONet, Branch-sharing	2355 (3.2)	767 MB (5.1)	4
PI-DeepONet, Trunk-sharing	1229 (1.6)	764 MB (5.1)	4

2.4. Reference solutions for validation

To evaluate the prediction accuracy of PI-DeepONet in both Case 1 and Case 2, we use reference solutions obtained from OpenFOAM [27, 28], an open-source computational fluid dynamics (CFD) toolbox widely used for simulating incompressible flows. The simulations are carried out using a steady-state solver based on the SIMPLE algorithm. Structured meshes are generated with `blockMesh`, and the mesh resolution is chosen to approximately match the spatial distribution of collocation points used in PI-DeepONet training. All simulations are run to convergence, and the resulting flow fields are verified to be smooth and physically consistent.

3. Results

3.1. Case 1: Reynolds number variation

3.1.1. Impact of the Reynolds number training range

We first investigate how the choice of training range for the Reynolds number affects the predictive performance of PI-DeepONet. In Case A, the model is trained on 11 Reynolds numbers sampled uniformly from Re = 1 to 100. Fig. 7 illustrates the predicted and reference velocity and pressure fields at Re = 1, 50, and 100, all of which fall within the training range.

Among these, the prediction at Re = 50, which lies near the center of the training range, exhibits excellent agreement with the OpenFOAM reference. The velocity field captures the structure of the recirculation region with high fidelity, and the pressure gradient along the channel walls is well resolved. At both Re = 1 and 100, which correspond to the boundaries of the training range, the predictive performance of PI-DeepONet is somewhat degraded compared to the interior. While the overall trends are similar, minor differences can be observed in how the velocity and pressure fields are affected. At Re = 1, the predicted velocity field deviates noticeably from the reference, particularly in the location of the recirculation vortex.

The extrapolation capability of PINN and PI-DeepONet is assessed in Fig. 8, which presents results at Re = 200, well outside the training range. While the overall flow structure is partially captured, both models exhibit noticeable performance degradation in this extrapolation regime. The deterioration is especially pronounced in the pressure field. In terms of velocity magnitude, PI-DeepONet demonstrates slightly better accuracy than PINN, although the improvement remains modest. These results indicate the difficulty of extrapolation beyond the training range, which remains a common challenge in operator learning for nonlinear flow problems.

To examine the sensitivity of model performance to the number of training points, we conducted an additional experiment using a reduced training set of only 6 Reynolds numbers, subsampled from the original 11-point configuration spanning Re = 1 to 100. The model maintained nearly the same level of performance at Re = 1, 50, and 100, suggesting a certain degree of robustness to the resolution of parametric sampling. However, when the number of training points was further reduced, the predictions deteriorated significantly, particularly near the boundaries,

indicating that a minimum sampling density is necessary to maintain interpolation quality.

Fig. 9 presents results at Re = 200, using a model trained on Reynolds numbers ranging from Re = 100 to 400 (Case B). In this case, the model accurately reproduces both the velocity and pressure fields throughout the domain. Compared to the prediction shown in Fig. 8 (Case A), the recirculation region and pressure contours are significantly improved, highlighting the importance of covering the target parameter range during training.

These observations collectively suggest that PI-DeepONet generalizes well within the training range, particularly near its center, while its accuracy degrades in extrapolation regimes. The deterioration observed at Re = 200 is primarily attributable to insufficient coverage of the parameter space rather than to a fundamental limitation of the network architecture. The sensitivity analysis further indicates that both the extent of the training range and the density of sampling points play critical roles in maintaining predictive accuracy. In principle, extrapolation accuracy could be improved either by extending the training range to include higher Reynolds numbers or by applying physics-informed fine-tuning at target conditions. However, changing the Reynolds number significantly alters the flow characteristics and residual distribution, which may require adapting the collocation strategy and optimization settings. Since the present study focuses on generalization under a fixed training setup and fast surrogate evaluation without case-specific retraining, a systematic investigation of such refinement strategies is left for future work.

To further assess the model's ability to capture physically meaningful flow characteristics, we compare the predicted length of the recirculation region against reference data in Fig. 10. This length is defined as the horizontal distance from the step to the reattachment point at the bottom wall, extracted from the predicted velocity field for each Reynolds number. The results show that PI-DeepONet accurately reproduces the recirculation length across a wide range of Re, with close agreement to both OpenFOAM simulations and prior numerical study by Biswas et al. [24]. The agreement is especially strong in the intermediate training range (Re = 25–75), while minor deviations are observed near the endpoints. These findings quantitatively support the qualitative comparisons presented earlier and demonstrate that the model captures salient flow features beyond pointwise accuracy.

3.1.2. Effect of network architecture

Next, we investigate how the choice of network architecture affects the predictive performance and computational efficiency of the model. Among the models in Fig. 3, both PINN-based architectures exhibit a noticeable drop in accuracy at Re = 1, even though this value lies within the training range, as illustrated in Fig. 11. The velocity fields predicted by PINN and Split-output PINN display spurious oscillations near the inlet and fail to reproduce the recirculation zone accurately. We attribute this to the direct concatenation of Re with the spatial coordinates (x, y, Re), which may hinder the network's ability to capture physics dominated by viscous effects at low Reynolds numbers. Interestingly, as shown in Fig. 12(a), the training loss histories for PINN and PI-DeepONet converge to similar total loss values, suggesting that both models achieve comparable minimization of the loss function. Furthermore, Fig. 12(b) shows that the relative contributions of the PDE and boundary terms remain balanced throughout training, with no strong dominance of either component in the late training stage. These observations suggest that the discrepancy in prediction accuracy is unlikely to be primarily caused by loss imbalance or incomplete convergence. Instead, it is more plausibly attributable to architectural inductive biases, such as the separation of input domains in DeepONet, which may play an important role in capturing relevant physical behaviors.

Despite these advantages, PI-DeepONet introduces additional memory and computational overhead. Since separate DeepONet models are required for each of the three outputs (u, v, and p), the number of fully connected neural networks (FNNs) increases from one (in PINN) to six, resulting in a significant rise in GPU memory usage and training time.

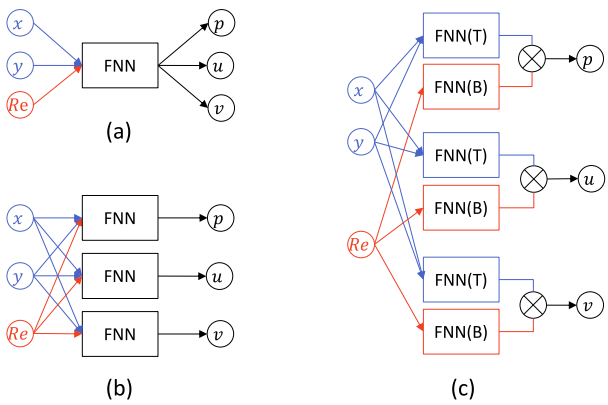

Fig. 3. Network architecture variants considered for incorporating the Reynolds number (Re) into the PINN framework. (a) A simple extension of the base PINN architecture (Fig. 1), where Re is directly appended to the spatial input coordinates (x, y) and fed into a single feedforward neural network (FNN) to predict all flow variables simultaneously. (b) A modular architecture in which three separate FNNs are used to predict the velocity components u, v, and pressure p, respectively. Each network receives the same input (x, y, Re) and shares an identical architecture. This configuration was also considered to enable a direct comparison with the DeepONet-based design in (c), where multiple sub-networks are similarly employed. (c) A DeepONet-based architecture, in which the input is split into two parts: spatial coordinates (x, y) for the trunk network (FNN(T)), and the Reynolds number Re for the branch network (FNN(B)). The output is computed as the inner product between the outputs of the trunk and branch networks, denoted by the symbol ⊗. Since one trunk-branch pair yields a single output, separate trunk-branch pairs are employed to predict u, v, and p individually.

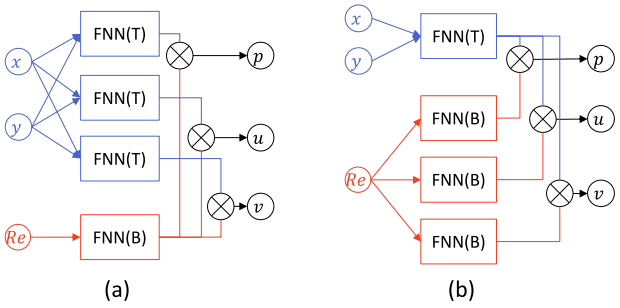

Fig. 4. Variants of the DeepONet architecture designed to reduce memory and computational costs by sharing network components. (a) Branch-sharing configuration: a single branch network processes the Reynolds number Re, and its output is shared across three separate trunk networks corresponding to u, v, and p. (b) Trunk-sharing configuration: a single trunk network processes the spatial coordinates (x, y), and its output is shared across three separate branch networks corresponding to u, v, and p. Both configurations aim to reduce the number of trainable parameters and memory usage, while maintaining the ability to learn distinct mappings for each physical quantity.

A detailed comparison of training time, GPU memory usage, and the number of FNNs for each architecture is provided in Table 2. The predictive accuracy of all variants was found to be comparable in our experiments, and is therefore omitted from further discussion. The training time and the GPU memory consumption were measured over 10,000 training epochs using `PyTorch MemLab`, which provides fine-grained tracking of tensor allocation and reuse. This table highlights the trade-off between model complexity and computational resources. As expected, the original PI-DeepONet model, comprising six FNNs, consumes the most memory (1.12 GB) and takes the longest training

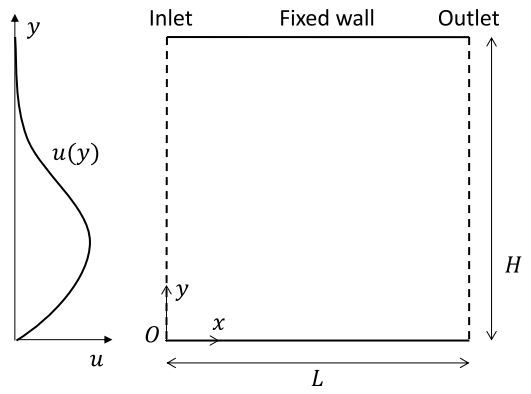

Fig. 5. Schematic illustration of the computational domain for the channel flow problem. This configuration is used to evaluate the model's ability to generalize across varying boundary conditions imposed at the inflow boundary. An example inlet velocity profile is shown in the figure for reference.

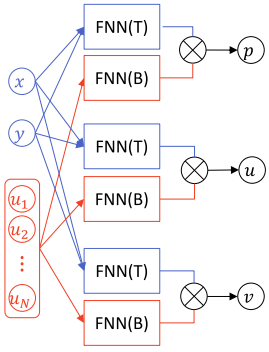

Fig. 6. DeepONet architecture for channel flow prediction under varying inlet velocity profiles (Case 2). The branch network takes as input a discretized velocity profile sampled at sensor locations along the inlet boundary, denoted as $\{u_i\}_{i=1}^{N}$, where each u_i represents the streamwise velocity at the ith sensor point. The trunk network receives the spatial coordinates (x, y), and the final output is obtained by taking the inner product of the two network outputs.

time (2569 s). In contrast, the baseline PINN with a single FNN uses only 149 MB and completes training in 725 s. The memory usage increases nearly linearly with the number of FNNs across all configurations, confirming that network duplication is the primary cause of memory cost.

Notably, the two DeepONet variants with shared networks reduce the FNN count to four, and accordingly, their memory usage drops to approximately 760 MB. However, their training times diverge: while the branch-sharing model takes almost as long as the original, the trunk-sharing model completes in roughly half the time. This discrepancy motivates further discussion in the following paragraph.

The observed trends in memory usage and training time can be partially explained by the underlying structure of the architectures. The memory consumption scales approximately with the number of independent FNNs, which is expected given that each network maintains its own set of parameters and activations. This accounts for the reduced memory usage observed in both the branch- and trunk-sharing variants of PI-DeepONet.

However, the training time does not follow a similar trend. While parameter sharing reduces the total number of learnable parameters, it does not necessarily reduce the computational workload. This is because the majority of training time is spent on backpropagation, where the same shared parameters are updated multiple times, once for each output variable that depends on them. As a result, the computational cost per epoch remains high, even with shared networks.

The trunk-sharing model achieves a substantial reduction in training time, completing training approximately 52% faster than the original PI-DeepONet, even though it shares the same number of parameters as the branch-sharing variant. At present, the exact reason for this discrepancy is not fully understood. We hypothesize that internal mechanisms in PyTorch, such as the reuse of the forward pass or more efficient gradient aggregation for shared modules, may contribute to this behavior. However, a rigorous understanding would require in-depth analysis of PyTorch's computational graph and autograd engine, which is beyond the scope of the current work. We regard this as an interesting direction for future investigation.

3.2. Case 2: Inlet condition variation

3.2.1. Assessment on in- and out-of-distribution boundary profiles

To investigate the generalization capability of PI-DeepONet under varying inlet conditions, we evaluate the model on two analytically defined velocity profiles that were not used during training. These test cases are selected to represent two distinct categories: one that lies close to the distribution of the training data, and another that deviates significantly in structure. This classification allows us to assess the model's performance both within and beyond the support of the learned function space.

The first test case is a parabolic profile defined as

$$u_{\text{in}}(\hat{y}) = 4U_{\max}\hat{y}(1-\hat{y}),$$

which corresponds to the fully developed laminar flow in a channel. Although this profile was not explicitly included in the training data, its one-peak symmetric shape is well aligned with the smooth RBF-based functions used during training. It is therefore treated as a representative in-distribution case.

The second profile is a cubic function given by

$$u_{\text{in}}(\hat{y}) = \frac{27}{4}U_{\max}\hat{y}(1-\hat{y})^2,$$

which features a flatter center and asymmetric curvature near the walls. This structural difference from the training samples makes it a suitable representative of an out-of-distribution input. In this test, both the parabolic and cubic profiles are normalized such that their maximum velocity is $U_{\max} = 1$.

Fig. 13 shows the comparison between the predicted and reference solutions for the parabolic profile. PI-DeepONet accurately reproduces both the velocity and pressure fields, with only minor discrepancies observed near the inlet. This confirms that the model generalizes well within the class of functions it has been exposed to during training.

Fig. 14 presents the results for the cubic profile. Despite its functional deviation from the training data, the model successfully captures the essential flow characteristics, including the location of the recirculation zone and the overall pressure distribution. This demonstrates the robustness of PI-DeepONet when applied to functionally different yet smooth boundary conditions.

The use of $U_{\max} = 1$ for evaluating test profiles was motivated by the distribution of maximum values observed in the training dataset. As shown in Table 1, this value lies just below the training mean and well within one standard deviation, making it a representative input magnitude. By contrast, a supplementary test using profiles scaled to $U_{\max} = 1.5$ corresponds to a value just beyond one standard deviation above the mean. Although still within the global maximum range, such profiles are underrepresented in the training data and may include steeper gradients and higher energy. This helps explain the observed degradation in prediction accuracy, indicating that the model's generalization capacity is somewhat sensitive to amplitude scaling beyond the typical training range.

3.2.2. Generalization to localized jet profiles

To further evaluate the model's generalization capability, we test PI-DeepONet on jet-type inlet velocity profiles characterized by strong

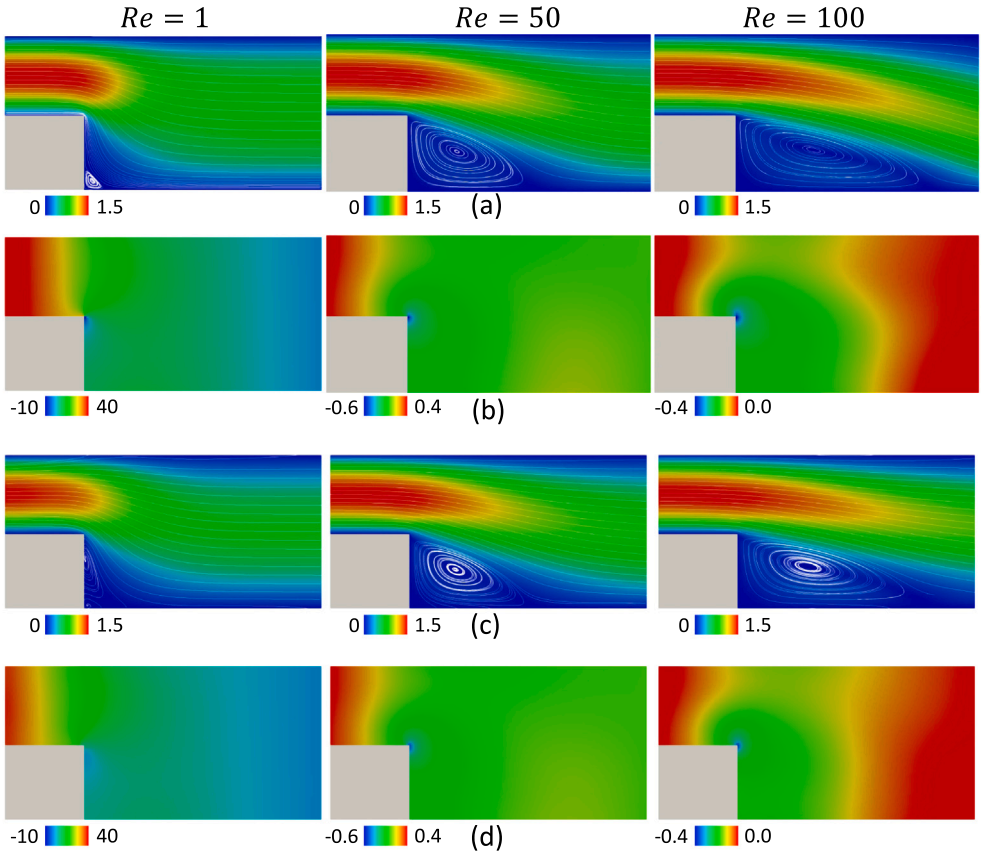

Fig. 7. Comparison of predicted and reference flow fields at Re = 1, 50, and 100 using PI-DeepONet trained on 11 Reynolds numbers sampled from Re = 1 to 100 (Case A). (a, b): Spatial distributions of velocity magnitude and pressure fields obtained from OpenFOAM. (c, d): Corresponding predictions obtained using PI-DeepONet. Excellent agreement is observed at Re = 50, near the center of the training range, while moderate deviations appear at the endpoints.

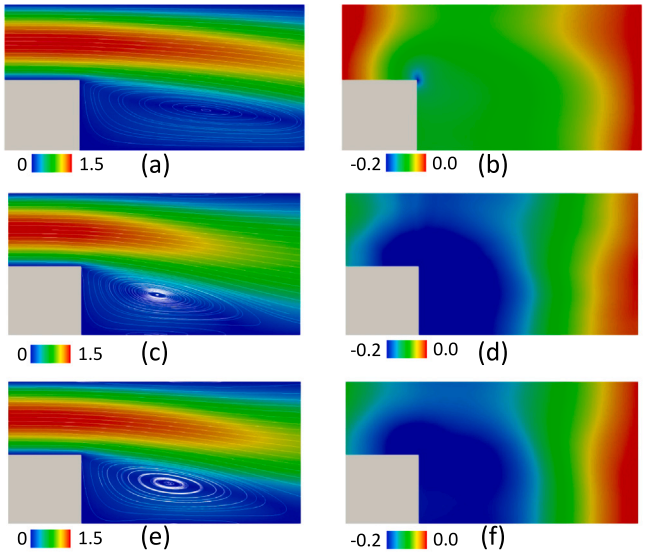

Fig. 8. Extrapolation performance at Re = 200 using models trained on Re = 1 to 100 (Case A). (a, b): Spatial distributions of velocity magnitude and pressure fields obtained from OpenFOAM. (c, d): Corresponding predictions obtained using PINN. (e, f): Corresponding predictions obtained using PI-DeepONet. Both models exhibit noticeable discrepancies, indicating the inherent difficulty of extrapolation beyond the training range.

spatial localization. Each profile is constructed using a single Gaussian radial basis function (RBF), resulting in a sharply peaked distribution centered at a specified location $\hat{y}_c$. The inlet velocity is defined as

$$u_{\text{in}}(\hat{y}) = 4U_0 \exp\left(-\frac{(\hat{y}-\hat{y}_c)^2}{2l^2}\right) \cdot \hat{y}(1-\hat{y}),$$

where U_0 and l denote the velocity and length scale parameters fixed during training, and $\hat{y}_c$ specifies the center position of the jet. Although the jet profile is defined analytically using a single Gaussian RBF, the resulting velocity distribution is sampled at the same predefined inlet sensor locations, and these sampled values are provided to the branch network. This is consistent with the training procedure described in Section 2.3.3.

Figs. 15 and 16 present results for jet positions at $\hat{y}_c = 0.2$ and $\hat{y}_c = 0.5$, respectively. For each case, the streamwise velocity u, wall-normal velocity v, and pressure p are compared against the OpenFOAM reference solution, together with the corresponding absolute error contours. The results show that PI-DeepONet accurately captures the overall flow structures induced by the localized jet, while the error contours highlight only minor discrepancies in limited regions.

While the present test focuses on predictive accuracy, the structure of the PI-DeepONet model also lends itself to practical applications such as design optimization and control. For instance, one could envision a scenario in which the optimal placement of a jet, acting as an actuator, is sought to achieve a desired flow pattern. Although the present example is simplified and not directly connected to such real-world applications, the ability to treat the jet position as a continuous input variable demonstrates the potential of PI-DeepONet for rapid parametric studies and preliminary design exploration.

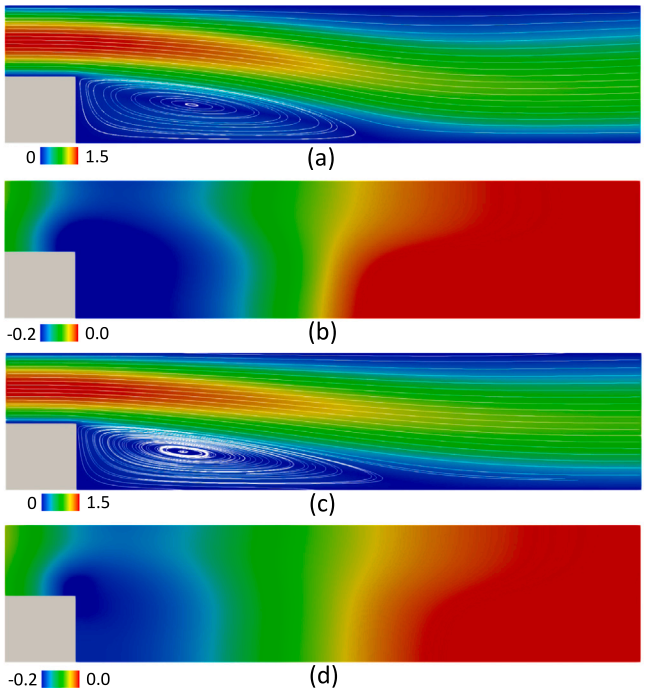

Fig. 9. Prediction at Re = 200 using PI-DeepONet trained on Re = 100 to 400 (Case B), where the target Re lies near the center of the training range. (a, b): Spatial distributions of velocity magnitude and pressure fields obtained from OpenFOAM. (c, d): Corresponding predictions obtained using PI-DeepONet. Compared to Fig. 8, the recirculation zone and pressure distribution are captured more accurately, demonstrating the effect of the training range.

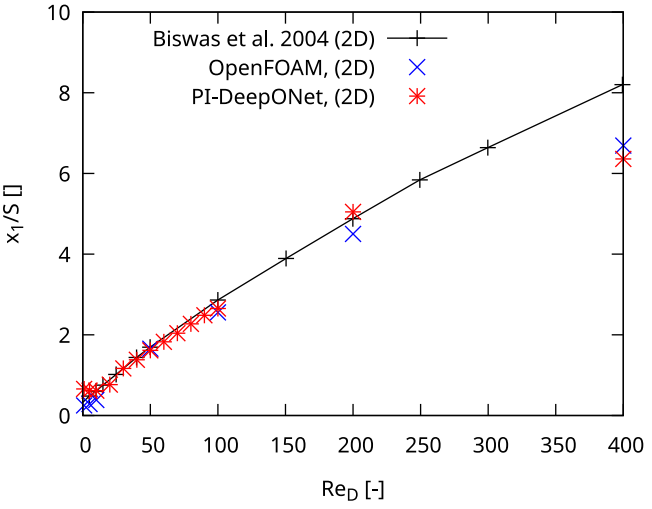

Fig. 10. Comparison of the recirculation region length as a function of Reynolds number. The results obtained by PI-DeepONet are compared with those from OpenFOAM simulations and existing reference data [24].

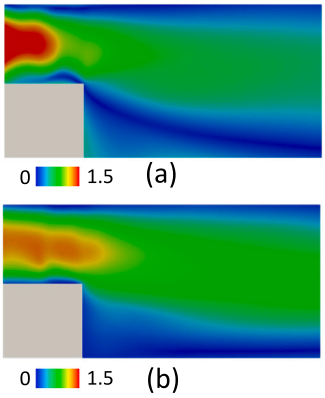

Fig. 11. Predicted velocity fields at Re = 1 obtained by (a) PINN and (b) Split-output PINN. Compared to Fig. 7(a) and (c), both models exhibit spurious oscillations near the inlet and fail to capture the recirculation zone accurately.

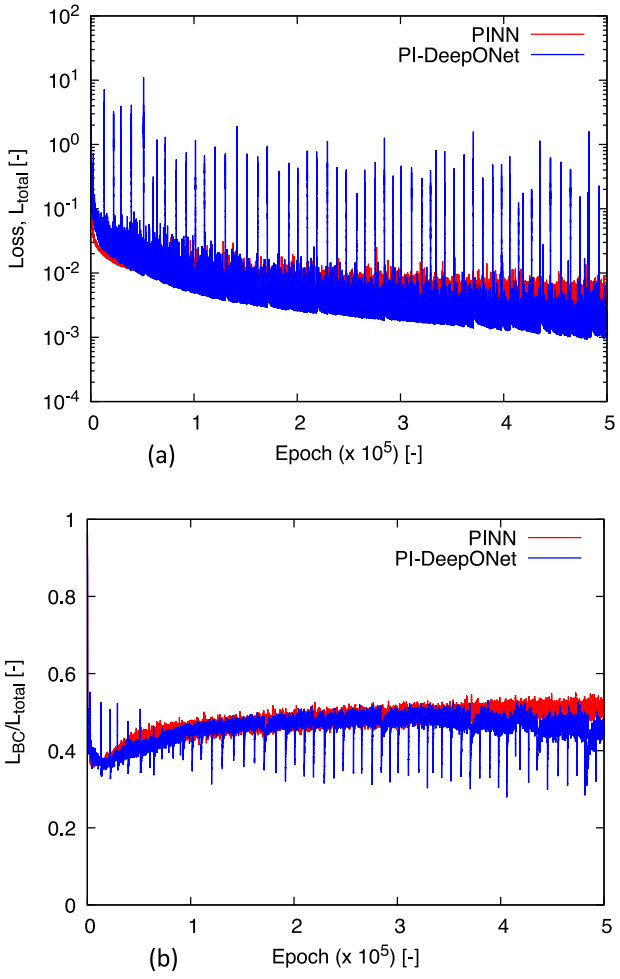

Fig. 12. Training loss histories for different network architectures applied to the backward-facing step flow. (a) Total loss $\mathcal{L}_{\text{total}}$ (including PDE residuals and boundary conditions) plotted over epochs for PINN and PI-DeepONet. (b) Boundary-loss ratio $\mathcal{L}_{\text{BC}}/\mathcal{L}_{\text{total}}$, indicating the balance between PDE residual minimization and boundary enforcement. The ratio curves in (b) are smoothed using a moving average with a window size of 101 epochs for visualization purposes.

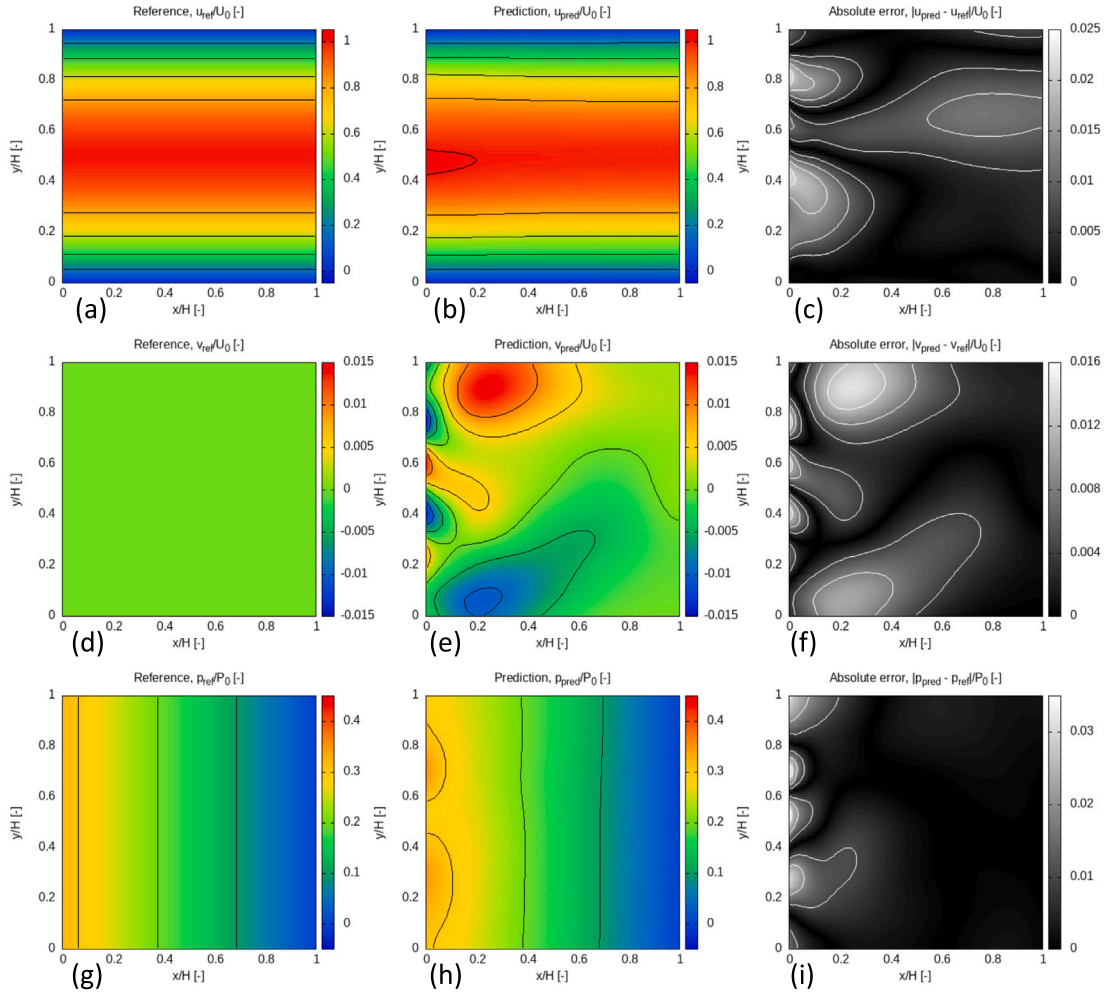

Fig. 13. Comparison of steady-state flow variables for the channel flow with a parabolic inlet velocity profile. Rows correspond to the streamwise velocity component u, wall-normal velocity component v, and pressure p, respectively. Columns show the analytical reference solution, the PI-DeepONet prediction, and the absolute error. The results demonstrate that the predicted fields closely match the reference solutions, with discrepancies primarily localized near the inlet region, where the boundary condition is given as input to the model.

4. Conclusion

In this study, we applied and evaluated the PI-DeepONet architecture as a surrogate modeling tool for incompressible flows under varying Reynolds numbers and boundary conditions. By combining the operator-learning capability of DeepONets with the physical constraints imposed by PINNs, the framework enables data-efficient learning without relying on labeled flow field data.

To demonstrate the effectiveness of the proposed framework, we applied it to two representative benchmark problems, each designed to reflect a distinct type of input variation: (1) backward-facing step flow with varying Reynolds numbers, and (2) channel flow with varying inlet velocity profiles. In both cases, the model demonstrated the ability to generalize beyond the training conditions, accurately reproducing velocity and pressure fields for unseen configurations. Notably, the model exhibits strong flexibility in handling arbitrary inlet velocity distributions without retraining, providing a clear advantage over conventional PINN-based and numerical approaches.

Beyond predictive accuracy, we also investigated how architectural design choices affect the model's performance, training dynamics, and computational cost. We found that while PI-DeepONet significantly enhances generalization compared to conventional PINNs, it requires substantially more memory and training time due to its use of multiple networks. To mitigate these costs, we explored shared-network variants of PI-DeepONet. Our results showed that memory usage decreases nearly in proportion to the number of neural networks, and that trunk sharing, in particular, leads to surprisingly significant reductions in training time, despite theoretical expectations to the contrary. This suggests that practical performance is influenced not only by model design but also by implementation-level factors such as framework-level optimization behavior.

Importantly, we found that comparable training losses across different architectures did not necessarily translate to similar generalization performance. This discrepancy highlights the critical role of architectural inductive bias in physics-informed learning, especially for models that rely on weak or indirect supervision. These findings underscore a broader implication: minimizing the training loss alone is not sufficient when designing robust and generalizable surrogate models for physical systems. Instead, careful consideration of the model's internal structure and its alignment with the underlying physics is essential for achieving consistent and trustworthy predictions.

Despite these promising results, the current utility of PI-DeepONet remains limited to relatively narrow parametric regimes and near-linear flow behaviors. In such scenarios, where only a small number

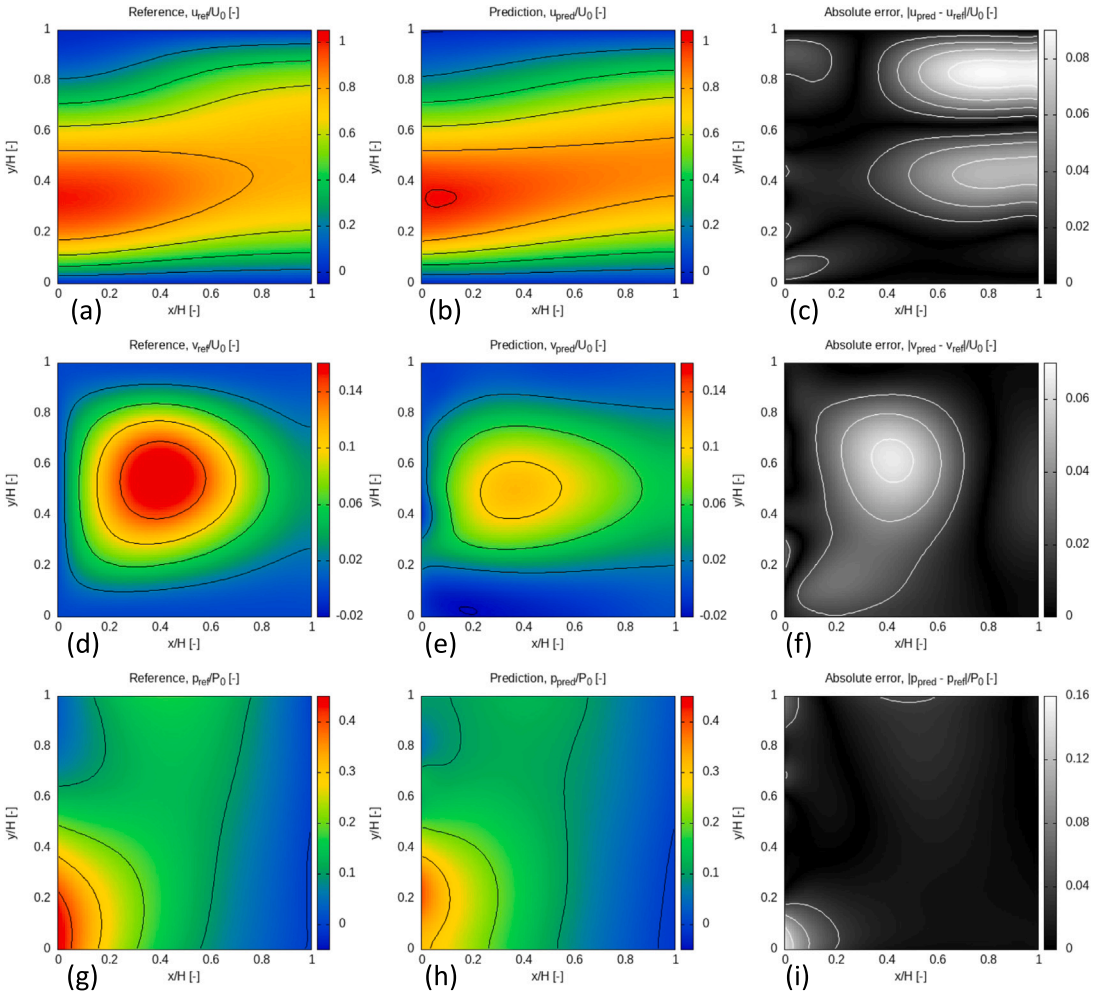

Fig. 14. Comparison of steady-state flow variables for the channel flow with a cubic (third-order polynomial) inlet velocity profile. Rows correspond to the streamwise velocity component u, wall-normal velocity component v, and pressure p, respectively. Columns show the OpenFOAM reference solution, the PI-DeepONet prediction, and the absolute error. PI-DeepONet successfully captures the overall structure of the flow and pressure distribution, showing good agreement with OpenFOAM throughout the computational domain.

of flow configurations are required, the computational advantage over conventional CFD solvers may not be realized, as the upfront training cost cannot be justified in the absence of repeated predictions. To fully realize the potential of operator learning in fluid mechanics, future research should focus on extending the model's representational capacity, improving training stability, and enabling scalability to more complex, nonlinear, and unsteady flow phenomena.

We believe that the present study provides valuable insights for advancing the development of reliable and generalizable surrogate models for scientific computing.

CRediT authorship contribution statement

Junya Onishi: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Harutaka Kitagawa:** Visualization, Validation, Software, Investigation, Formal analysis, Data curation. **Rishabh Puri:** Writing – review & editing, Validation, Methodology. **Mario Rüttgers:** Writing – review & editing, Validation, Methodology. **Rakesh Sarma:** Writing – review & editing, Methodology, Investigation. **Andreas Lintermann:** Writing – review & editing, Resources, Project administration. **Makoto Tsubokura:** Writing – review & editing, Supervision, Resources, Project administration.

Declaration of competing interest

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

Acknowledgments

This work was partly supported by the NCSA-Inria-ANL-BSC-JSC-Riken-UTK Joint Laboratory for Extreme Scale Computing (JLESC, https://jlesc.github.io/). Additional support was provided by JSPS KAKENHI Grant Number 25K15154, the German Research Foundation through the Walter Benjamin Fellowship RU 2771/1-1, and the HANAMI project within the European Union Horizon Europe Programme - Grant Agreement Number 101136269 under the call HORIZON-EUROHPC-JU-2022-INCO-04. Portions of the computations were conducted using the FUJITSU Supercomputer PRIMEHPC FX1000 and the FUJITSU Server PRIMERGY GX2570 (Wisteria/BDEC-01) at the Information Technology Center, The University of Tokyo. The authors would like to express their sincere gratitude to all funding organizations and computational facilities involved in this study.

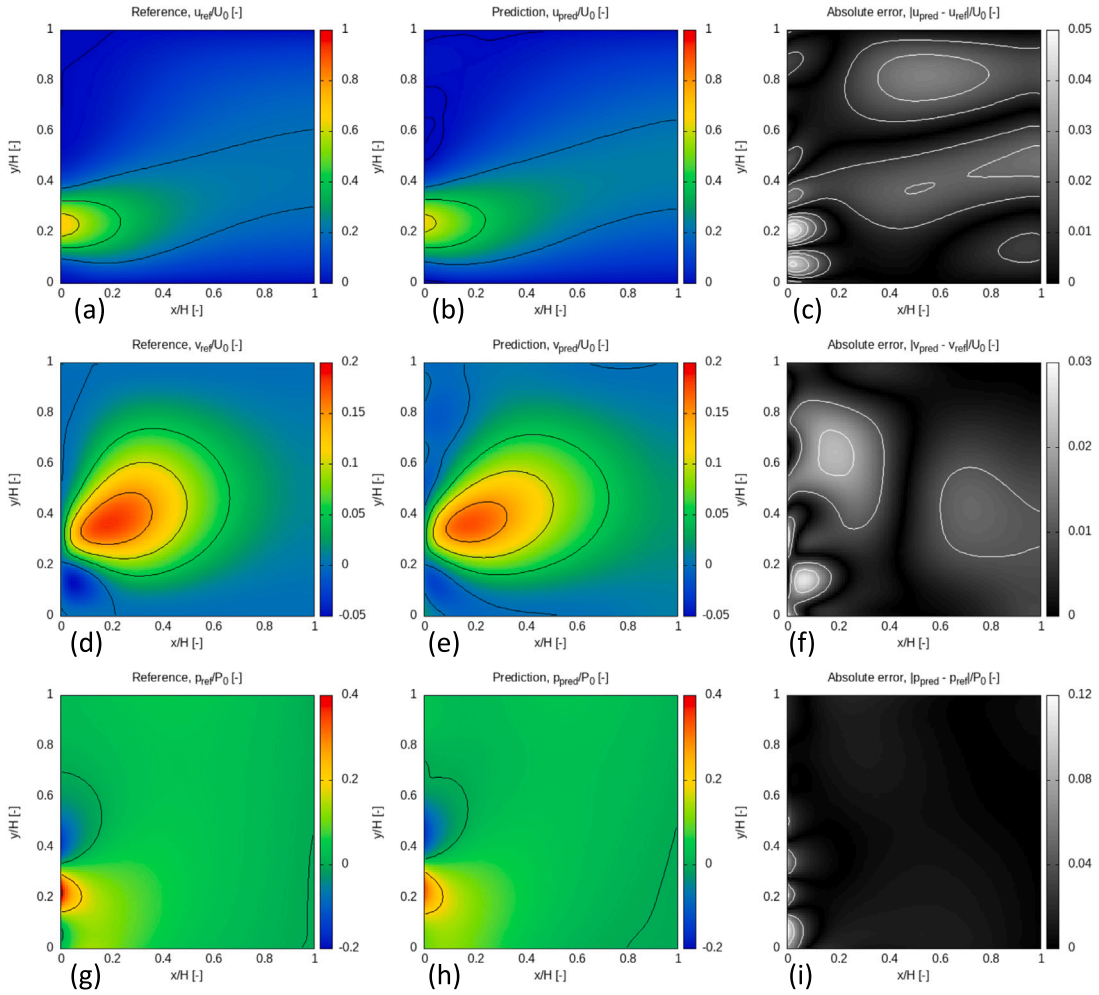

Fig. 15. Comparison of steady-state flow variables for the channel flow with a jet-type inlet velocity profile located at $y/H = 0.2$. Rows correspond to the streamwise velocity component u, wall-normal velocity component v, and pressure p. Columns show the OpenFOAM reference solution, the PI-DeepONet prediction, and the absolute error.

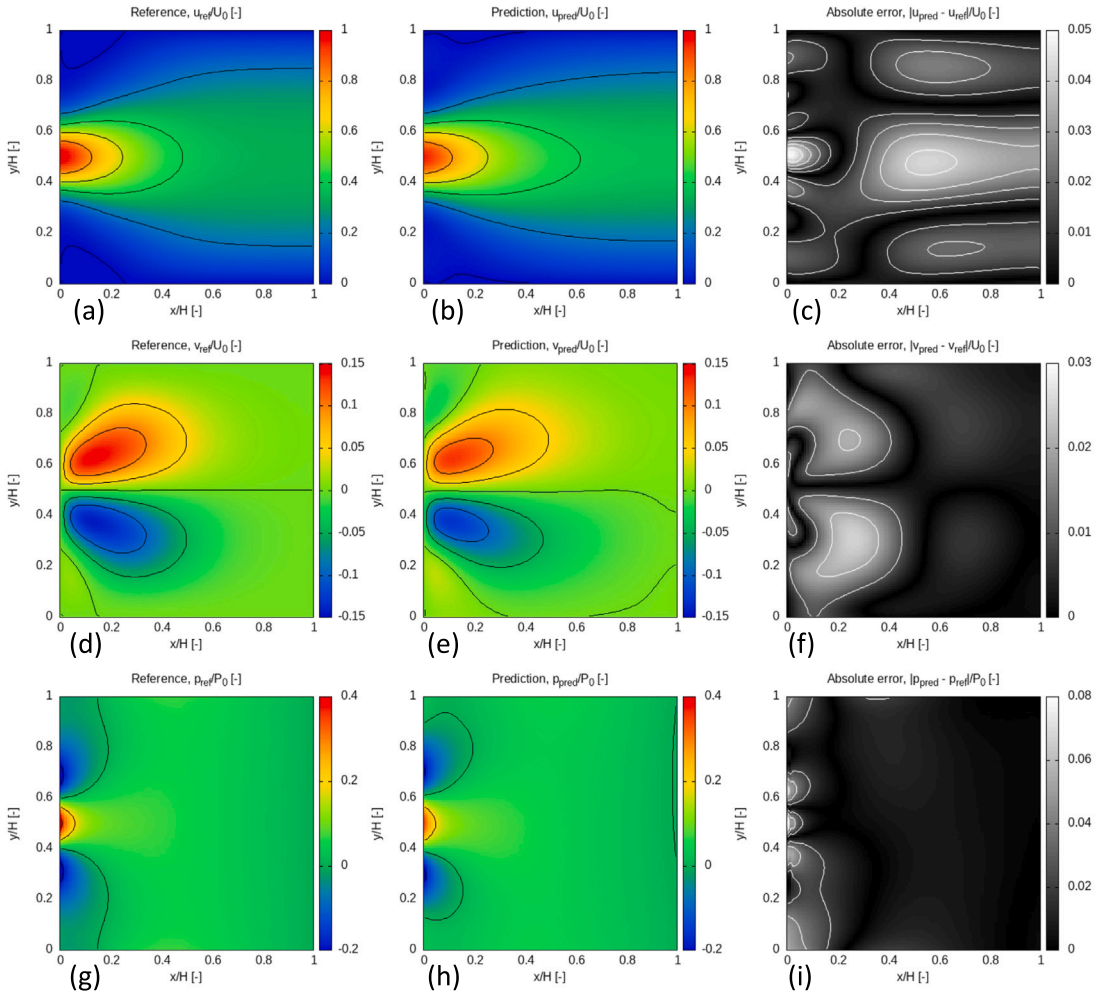

Fig. 16. Comparison of steady-state flow variables for the channel flow with a jet-type inlet velocity profile located at $y/H = 0.5$. Rows correspond to the streamwise velocity component u, wall-normal velocity component v, and pressure p. Columns show the OpenFOAM reference solution, the PI-DeepONet prediction, and the absolute error.

References

[1] Berkooz G, Holmes P, Lumley JL. The proper orthogonal decomposition in the analysis of turbulent flows. Annual Rev Fluid Mech 1993;25(1):539–75.

[2] Holmes P. Turbulence, coherent structures, dynamical systems and symmetry. Cambridge University Press; 2012.

[3] Rouizi Y, Favennec Y, Ventura J, Petit D. Numerical model reduction of 2D steady incompressible laminar flows: Application on the flow over a backward-facing step. J Comput Phys 2009;228(6):2239–55.

[4] Hesthaven JS, Ubbiali S. Non-intrusive reduced order modeling of nonlinear problems using neural networks. J Comput Phys 2018;363:55–78.

[5] Karniadakis GE, Kevrekidis IG, Lu L, Perdikaris P, Wang S, Yang L. Physics-informed machine learning. Nature Rev Physics 2021;3(6):422–40.

[6] Willard J, Jia X, Xu S, Steinbach M, Kumar V. Integrating physics-based modeling with machine learning: A survey. 1, (1):2020, p. 1–34, arXiv preprint arXiv:2003.04919.

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

[8] Jin X, Cai S, Li H, Karniadakis GE. NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. J Comput Phys 2021;426:109951.

[9] Puri R, Onishi J, Rüttgers M, Sarma R, Tsubokura M, Lintermann A. On the choice of physical constraints in artificial neural networks for predicting flow fields. Future Gener Comput Syst 2024;161:361–75.

[10] Eivazi H, Tahani M, Schlatter P, Vinuesa R. Physics-informed neural networks for solving Reynolds-averaged Navier–Stokes equations. Phys Fluids 2022;34(7):075117.

[11] Mao Z, Jagtap AD, Karniadakis GE. Physics-informed neural networks for high-speed flows. Comput Methods Appl Mech Eng 2020;360:112789.

[12] Kissas G, Yang Y, Hwuang E, Witschey WR, Detre JA, Perdikaris P. Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks. Comput Methods Appl Mech Eng 2020;358:112623.

[13] Qiu R, Huang R, Xiao Y, Wang J, Zhang Z, Yue J, Zeng Z, Wang Y. Physics-informed neural networks for phase-field method in two-phase flow. Phys Fluids 2022;34(5):052109.

[14] Cai S, Mao Z, Wang Z, Yin M, Karniadakis GE. Physics-informed neural networks (PINNs) for fluid mechanics: a review. Acta Mech Sinica 2021;37:1727–38.

[15] Lu L, Jin P, Pang G, Zhang Z, Karniadakis GE. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nature Mach Intelligence 2021;3(3):218–29.

[16] Li Z-Y, Kovachki NB, Azizzadenesheli K, Liu B, Bhattacharya K, Stuart AM, Anandkumar A. Fourier neural operator for parametric partial differential equations. In: Int conf learn represent. 2020, abs/2010.08895.

[17] Wang S, Wang H, Perdikaris P. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. Sci Adv 2021;7(40):eabi8605.

[18] Wang S, Wang H, Perdikaris P. Improved architectures and training algorithms for deep operator networks. J. Sci Comput 2021;92(2):35.

[19] Mandl L, Goswami S, Lambers L, Ricken T. Separable physics-informed DeepONet: Breaking the curse of dimensionality in physics-informed machine learning. Comput Methods Appl Mech Eng 2025;434:117586.

[20] Goswami S, Bora A, Yu Y, Karniadakis GE. Physics-informed deep neural operator networks. In: Computational methods in engineering & the sciences. Cham: Springer International Publishing; 2023, p. 219–54.

[21] Zhang J, Zhang S, Shen J, Lin G. Energy-dissipative evolutionary deep operator neural networks. J Comput Phys 2024;498:112638.

[22] Li W, Bazant MZ, Zhu J. Phase-field DeepONet: Physics-informed deep operator neural network for fast simulations of pattern formation governed by

gradient flows of free-energy functionals. Comput Methods Appl Mech Eng 2023;416:116299.
[23] Griewank A. Evaluating derivatives : principles and techniques of algorithmic differentiation: Principles and techniques of algorithmic differentiation, second edition. 2nd ed.. Cambridge, TAS, Australia: Cambridge University Press; 2008.
[24] Biswas G, Breuer M, Durst F. Backward-facing step flows for various expansion ratios at low and moderate Reynolds numbers. J Fluids Eng 2004;126(3):362–74.
[25] He J, Koric S, Abueidda D, Najafi A, Jasiuk I. Geom-DeepONet: A point-cloud-based deep operator network for field predictions on 3D parameterized geometries. Comput Methods Appl Mech Eng 2024;429:117130.
[26] He J, Kushwaha S, Park J, Koric S, Abueidda D, Jasiuk I. Predictions of transient vector solution fields with sequential deep operator network. Acta Mech 2024;235(8):5257–72.
[27] Weller H, Tabor G, Jasak H, Fureby C. A tensorial approach to computational continuum mechanics using object-oriented techniques. Comput Phys 1998;12(6):620–31.
[28] OpenFOAM Foundation. The OpenFOAM foundation. 2024, https://openfoam.org. [Accessed: 23 April 2025].