Intelligent Computing
A SCIENCE PARTNER JOURNAL

RESEARCH ARTICLE

Stacked Deep Learning Models for Fast Approximations of Steady-State Navier–Stokes Equations for Low *Re* Flow

Shen Wang[1], Mehdi Nikfar[2], Joshua C. Agar[3*], and Yaling Liu[1,4*]

[1]Mechanical Engineering and Mechanics, Lehigh University, Bethlehem, PA, USA. [2]Biomedical Engineering, School of Medicine, Johns Hopkins University, Baltimore, MD 21205, USA. [3]Mechanical Engineering and Mechanics, Drexel University, Philadelphia, PA 19104, USA. [4]Bioengineering, Lehigh University, Bethlehem, PA, USA.

*Address correspondence to: jca92@drexel.edu (J.C.A.), yal310@lehigh.edu (Y.L.)

Citation: Wang S, Nikfar M, Agar JC, Liu Y. Stacked Deep Learning Models for Fast Approximations of Steady-State Navier–Stokes Equations for Low *Re* Flow. *Intell. Comput.* 2024;3:Article 0093. https://doi.org/10.34133/icomputing.0093

Submitted 20 July 2023
Accepted 28 March 2024
Published 26 June 2024

Computational fluid dynamics (CFD) simulations are broadly used in many engineering and physics fields. CFD requires the solution of the Navier–Stokes (N-S) equations under complex flow and boundary conditions. However, applications of CFD simulations are computationally limited by the availability, speed, and parallelism of high-performance computing. To address this, machine learning techniques have been employed to create data-driven approximations for CFD to accelerate computational efficiency. Unfortunately, these methods predominantly depend on large labeled CFD datasets, which are costly to procure at the scale required for robust model development. In response, we introduce a weakly supervised approach that, through a multichannel input capturing boundary and geometric conditions, solves steady-state N-S equations. Our method achieves state-of-the-art results without relying on labeled simulation data, instead using a custom data-driven and physics-informed loss function and small-scale solutions to prime the model for solving the N-S equations. By training stacked models, we enhance resolution and predictability, yielding high-quality numerical solutions to N-S equations without hefty computational demands. Remarkably, our model, being highly adaptable, produces solutions on a 512 × 512 domain in a swift 7 ms, outpacing traditional CFD solvers by a factor of 1,000. This paves the way for real-time predictions on consumer hardware and Internet of Things devices, thereby boosting the scope, speed, and cost-efficiency of solving boundary-value fluid problems.

Introduction

Many fluid problems are governed by nonlinear partial differential equations (PDEs) based on the Navier–Stokes (N-S) equations. Numerical simulations of these fluid dynamics problems require the resolution of N-S equations in a discretized spatial and temporal form. Various methods like the finite difference method (FDM), finite volume method (FVM) [1,2], Lattice–Boltzmann method (LBM) [3–5], and finite element method (FEM) [6–8] exist to solve the N-S equations. However, these methods can become computationally demanding and memory intensive when high-resolution meshes are needed. The challenge is further intensified by the complexity of determining suitable computational grids [9]. Although the preprocessing tasks can be somewhat alleviated by commercial software, they still necessitate a solid understanding of computational fluid dynamics (CFD) principles such as the governing equations, selection of appropriate flow solvers, upwind schemes, and turbulent models [10], and they have a negligible impact on the computational cost. Moreover, regardless of computational similarity, even subtle changes in the boundary conditions or geometric domain necessitate completely reworking the simulations.

The necessity for rapid, automated solutions to the N-S equations has surged in various fields for use in video game engines, ocean current or hurricane forecasts, prediction of oil spills or fire smoke spreading, and porous media flow, among others. These applications demand physics-informed dynamic control systems and reinforcement learning. Unfortunately, the current commercial software is inadequate for fully automated applications. Therefore, automating and accelerating real-time fluid simulation can enable new applications where speed, energy, and computational cost are mission-critical [11–16]. The rising interest lies in the development of precise, coherent, and reduced-order models capable of representing flow characteristics.

Machine learning (ML), particularly deep learning (DL), has made important strides in computational mechanics, serving as an approximation for dynamic spatiotemporal systems [17–21]. DL models are typically overparameterized models that can serve as data-driven approximations between the input and the target. By introducing "damaging mechanisms" through regularization, a loss function can be optimized to generalize within the distribution of the training data.

A purely data-driven approach utilizes the DL model as a "black box" to map input to output. However, generating a

comprehensive dataset to train robust DL models can be computationally costly and demands an in-depth understanding of the statistical distribution of the training dataset [22], and it is hard to achieve robustness when models are overparameterized and training data are limited [23]. Recently, physics-informed neural networks (PINNs) have been used to add physics constraints that improve the generalizability of DL models. For instance, physical laws including PDEs and initial and boundary conditions are explicitly embedded into the DL model. PINNs use physics as a parsimonious regularizer that enhances the robustness and interpretability of the DL model [22]. Trained PINNs can provide alternative solutions to PDEs under various boundary and initial conditions [24]. However, once the training procedure concludes, a trained model can only yield solutions for a specific configuration of inputs. It is therefore not applicable when the geometry, boundary, or initial conditions change. Additionally, trained models are not readily adjustable for system-level discrepancies, such as friction changes [25]. Consequently, these networks can only accelerate narrowly defined problems.

Several research groups have recently ventured into predicting the results of different PDEs by building PINNs [19,20,24,26–30]. For example, a weakly supervised algorithm was capable of solving Laplace's equations [28]. In this work, a fully convolutional encoder–decoder network in a U-Net architecture [31] was used to predict the solution. This study integrated physics-informed loss with a generative model [32–34] that retained the intrinsic relationships between neighboring nodal points by introducing a convolution kernel based on Laplace's equation. Without access to any solved simulation data, the trained model could predict the solution with different boundary conditions. Nevertheless, the applicability of this method was limited to the heat equation with Dirichlet boundary conditions. N-S equations, in contrast, are second-order nonlinear PDEs containing multiple equations and variables with convective, pressure, and viscous terms, and are much more complicated than heat equations. The heat equation is akin to only the viscous term in N-S equations; hence, this method cannot be readily applied to general N-S equations.

In this work, we focus on solving the N-S equations in 2-dimensional (2D) space. We developed a weakly supervised method that considers complicated momentum and continuity equations, the pressure field, and velocities in both dimensions. Our method generates steady-state solutions to flow problems governed by N-S equations in roughly 5 to 7 ms, without requiring pre-computed CFD results. We accomplished this by initiating warm-up through pre-run iterations or coarse solutions. Utilizing convolutional U-Nets, we accurately approximated the solutions of steady-state N-S equations under varying boundary conditions and with internal obstacles by training stacked models. To improve the performance, we imposed 2 types of constraints on the loss function: physics-informed constraints, represented by the residue of the equations, and data-driven constraints, represented by the differences between the output and the known values on the boundaries. We validated the models by comparing the results to FDM solutions. Our approach provided solutions with a root mean square error (RMSE) of 0.04 for velocity fields compared to the ground truth from FDM simulations while reducing computation time by a factor of 1,000.

Materials and Methods

Model architecture

To approximate solutions to the N-S equation with boundary conditions, we designed a neural network architecture in the form of a convolutional U-Net (Fig. 1). The U-Net is a modified autoencoder. Autoencoders are composed of an encoder, which learns a compact representation of the data, and a decoder, which reconstructs a target of similar dimensionality from this compact representation (Fig. 1). U-Nets include message passing from the

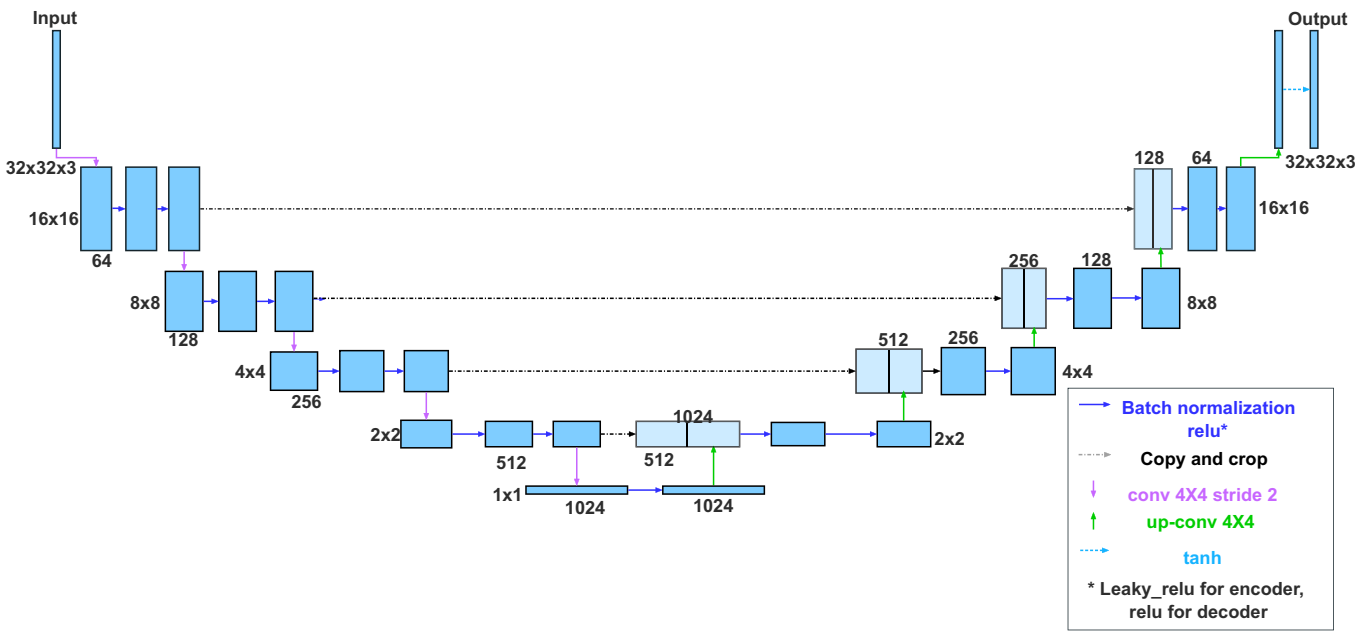

Fig. 1. Schematic representation of the architecture of the model. The input data can be sized differently. In this figure, the example input domain has a size of (32 × 32 × 3) for width, height, and number of channels (depth), respectively. The number of layers in both the encoder and decoder is equal to $\log_2 a$, where a is the dimension of the square matrix. In this figure, there are 5 layers. The width and height of the data decrease in the encoder and then increase until the original size is recovered in the decoder. Data representations in the same row have the same width and height, as annotated at the left or right, and the same depth, except where specifically noted above or below.

encoder to the decoder to assist in reconstructing high-resolution representations. U-Nets have been used for image reconstruction from the input with given constraints in different fields [35–37]. Fully convolutional networks have been used for DL-based CFD simulations as predicting models [32,33,38] and by generative adversarial networks (GANs) [32,34]. The designed U-Net is constructed from convolutional layers that take the input and compress the information until reaching the bottleneck layer. The output from the encoder is then passed to the decoder, which reconstructs the input to its original size and maps the output to the range $[-1,1]$. The output from the final layer is scaled via a linear mapping to match the expected magnitude for each channel, forming the final output of the model.

The fully convolutional network contains an encoder and decoder with convolutional blocks. Each convolutional block includes convolutional layers with a kernel size of (4, 4) and a stride of 2, batch normalization, and a leaky rectified linear unit (ReLU) activation layer defined by:

$$Leaky\_Relu(x) = \begin{cases} 0.2x & x \le 0 \\ x & x > 0 \end{cases}. \tag{1}$$

The decoder with deconvolutional layers takes the output of the preceding layer concatenated with the corresponding encoding layers. A multichannel outcome from the decoder followed by a hyperbolic tangent *tanh* activation layer is reconstructed to reach the original size of the input.

Physics-informed data-driven loss function

Optimization based on purely data-driven loss functions resulted in costs for preparing the numerical solutions as labeled data [33,38]. To improve the model performance with minimal low-cost computationally simulated data, we constructed a custom loss function that combined the physics-informed loss L_{phy}, which represents the physics equations and constraints defined by the residue of the computational operations, and the data-driven loss L_b, which represents the direct comparisons with the boundary conditions, to constrain the model. The physical model used in this study is based on 2D incompressible fluid dynamics constrained by N-S equations. The governing equations can be rewritten as:

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} - v\frac{\partial u}{\partial y} - \frac{\partial p}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right), \tag{2}$$

$$\frac{\partial v}{\partial t} = -u\frac{\partial v}{\partial x} - v\frac{\partial v}{\partial y} - \frac{\partial p}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right), \tag{3}$$

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = -\left(\left(\frac{\partial u}{\partial x}\right)^2 + 2\frac{\partial u}{\partial y}\frac{\partial v}{\partial x} + \left(\frac{\partial v}{\partial y}\right)^2\right). \tag{4}$$

The physics-informed loss functions are based on the residue of the PDEs, defined in the following equation in which N_I denotes the number of internal nodes:

$$L_{phy} = \frac{1}{N_I}\sum_{i=1}^{N_I}\left\|R\left(x^i, y^i\right)\right\|^2 + \lambda_N L_{Neumann}. \tag{5}$$

The residues of the PDEs (Eq. 6) consist of 3 sublosses of each equation in N-S equations:

$$R = \lambda_1 |L_{X-momentum}| + \lambda_2 |L_{Y-momentum}| + \lambda_3 |L_{Continuity}|. \tag{6}$$

Each subloss has subterms that represent the viscous term, convective term, and pressure term. For nonboundary nodes, assuming $\widehat{u} = \frac{1}{2}\left(u^n_{i+1,j} - u^n_{i-1,j}\right)$ and $\widehat{p} = \frac{1}{2}\left(p^n_{i+1,j} - p^n_{i-1,j}\right)$, the 3 subterms of the loss should be written as:

$$L_{X-Momentum}(\mathbf{W},\mathbf{b}) = \frac{1}{h^2}\frac{1}{Re}\sum_{i,j} Conv2d(K,U)_{ij} + \frac{1}{h}\sum_{i,j}\left(u_{i,j}\widehat{u}_{i,j} + v_{i,j}\widehat{u}_{i,j}\right) - \frac{1}{h}\widehat{P}, \tag{7}$$

$$L_{Y-Momentum}(\mathbf{W},\mathbf{b}) = \frac{1}{h^2}\frac{1}{Re}\sum_{i,j} Conv2d(K,V)_{ij} + \frac{1}{h}\sum_{i,j}\left(u_{i,j}\widehat{v}_{i,j} + v_{i,j}\widehat{v}_{i,j}\right) - \frac{1}{h}\widehat{P}, \tag{8}$$

$$L_{Continuity}(\mathbf{W},\mathbf{b}) = \frac{1}{4}\sum_{i,j} Conv2d(K,P)_{ij} + \sum_{i,j}\left(u_{i,j}\widehat{u}_{i,j} + 2\widehat{u}_{i,j}\widehat{v}_{i,j} + v_{i,j}\widehat{v}_{i,j}\right). \tag{9}$$

K denotes the physics-informed kernels similar to kernels reported in [28], which are operated on a domain that contributes to the loss of the Laplace operator corresponding to the viscous terms. *Conv2d* denotes the 2D convolutional operations, and h defines the discretization unit.

In our practice, K has 2 different scenarios. We extended the idea described in [28] for Neumann boundaries. For example, if a Neumann boundary is to be put on the left-hand side, K changes accordingly:

$$K = \begin{cases} \begin{pmatrix} 0 & 1/4 & 0 \\ 1/4 & -1 & 1/4 \\ 0 & 1/4 & 0 \end{pmatrix} & \text{for internal nodes,} \\ \begin{pmatrix} 0 & 1/4 & 0 \\ -1/2 & -1 & 1/2 \\ 0 & 1/4 & 0 \end{pmatrix} & \text{for Neumann boundary.} \end{cases} \tag{10}$$

Another term, $L_{Neumann}$, which represents the loss of any Neumann boundaries, is added in Eq. 5 as a physics-driven loss. Suppose that the physics domain has the N_X and N_Y nodes in the X and Y directions, respectively, and has N_N Neumann boundary nodes in total. For the boundaries that have Neumann boundary conditions, the following equation would apply for this loss:

$$L_{Neumann} = \frac{1}{N_N}\sum_{i=1}^{N_N}\left\|S\left(x_{0;N_Y}, y^i\right) - S\left(x_{1;N_{Y-1}}, y^i\right)\right\|^2 + \left\|S\left(x^i, y_{0;N_X}\right) - S\left(x^i, y_{1;N_{X-1}}\right)\right\|^2. \tag{11}$$

The total loss function also includes contributions from Dirichlet boundary conditions. Although the boundaries are not part of the output, they are considered in the loss function using an additional term. The data-driven loss is calculated from the mean squared error between the generated solution and the known values on the boundaries. The boundary loss represents the distances between the values generated by the model and the actual Dirichlet boundary conditions, which helps to reinforce the boundaries. Suppose $S = (u, v, p)$ to be physical variables that are subject to boundary conditions given by $\hat{S}$, where $\hat{S}$ denotes known values of the boundary at the corresponding domains, and N_b denotes the number of boundary nodes. For the boundaries that contain Dirichlet boundary conditions, the following equation would apply for this loss:

$$L_b = \frac{1}{N_b}\sum_{i=1}^{N_b}\left\|S\left(x_{0,N}, y^i\right) - \hat{S}\left(x_{0,N}, y^i\right)\right\|^2 + \left\|S\left(x^i, y_{0,N}\right) - \hat{S}\left(x^i, y_{0,N}\right)\right\|^2. \quad (12)$$

Finally, the total objective function is:

$$L = L_{phy} + \lambda_b L_b. \quad (13)$$

Data representation

To demonstrate the effectiveness of our method, we tested our model on solving the N-S equation in a 2D space encompassing 3 physical variables, i.e., the velocity vector in the x and y directions and the pressure field, by treating the data as a multichannel image. A single input data cube in this work is $I = (S; G)$, where the aforementioned S is the physical variables (u, v, p) in 2D space and each represents a channel. G is the optional channel corresponding to a binary geometry mask, where 1 denotes the regions of solid boundaries considered as known boundary conditions, while 0 stands for regular regions to be generated by the model. In summary, the DL model takes input I, which is a composite of 2D matrices with 4 channels, and generates solution $O = (u_{approx}, v_{approx}, p_{approx})$ correspondingly.

Training experiments

We trained a base model that has the capability to predict the flow problem. Then, we trained a series of succeeding models as an intermediate step and successfully acquired pre-trained models. We treated such trained models as the starting point for further training steps. Ideas for subsequent models include increasing complexity via random choices of the inlet, larger domain size, geometric configurations, etc. We validated the model by comparing its predicted solutions to the numerical solutions. Throughout this iterative training process, we employed a "jump-start" strategy utilizing "warm-up data," detailed in the "Warm-up data preparation" section, to facilitate more effective and efficient model learning. The results of the base and advanced models are described in the Results and Discussion section.

During training for each model, we followed the procedures outlined in Fig. 2. We assumed a predictable range of boundary conditions of the model to decide the overall scale of the output from the model with a split of 80% for training and 20% for testing (Fig. 2A). Using the developed DL framework, we trained the model (Fig. 2B) with the described total loss function (Eq. 13).

CFD solver

In our study, we utilized the lid-driven cavity flow problem as a benchmark scenario to validate our DL models. The numerical solver employed is designed around the FDM, a discretization technique that approximates derivatives in the N-S equations using differences between grid points. The lid-driven cavity flow was simulated on a uniform grid. For the solver formulations, the 2D incompressible N-S equations were used, discretized via central differences for spatial derivatives and a forward difference for the time derivative. The boundary conditions set for the base and advanced models can be seen in Fig. 3.

Transfer learning

During the training processes, we adopted the technique of transfer learning progressively. Transfer learning enables our models to commence training from an advanced stage, where not all parameters start with initialized values. Specifically, certain parameters are fine-tuned based on the optimized values inherited from predecessor models, rather than being trained from their initial states that are sampled randomly. This approach considerably enhances training efficiency and model performance. Detailed insights into our various applications of transfer learning are presented in the Results section.

Warm-up data preparation

Optimizing a single U-Net model to solve N-S equations was challenging when it was initialized randomly. To overcome this challenge, we used a stacked U-Net structure of increasing spatial resolution. Similarly, so-called Stack-GANs have been used to generate realistic high-resolution images from text [39,40].

We initialized the direct input of our model to contain computational domains marked by boundary conditions and unsolved nodes set to zeros. This "cold input" created a neutral starting point that allows the model to learn from fundamental principles dictated by the boundary conditions. During our experiments, we discovered that a systematic warm-up phase was crucial. Therefore, the initial stages of the models were exposed to a certain level of "pre-computed" data as input, facilitating more stable and effective weight updates. To do this, we employed a step-by-step training strategy (Fig. 4), starting with generating weakly supervised data with pre-run iterations or coarse analytical solutions, "warm-up data," as input. The warm-up data preparation involved generating weakly supervised data through pre-run iterations or coarse analytical solutions, considerably reducing computational costs. Specifically, the preparation of each set of training data required less than 2.5 s, thus serving as an efficient method to jump-start the training process. When the models demonstrated the ability to train effectively without warm-up data, we reverted to using "cold input" to ensure that the model gained the capability to process neutral inputs as initially intended.

The warm-up phase comprised 2 primary types of data to support the training: pre-run solutions and coarse solutions. Both of the datasets are 4D matrices in the size of $N \times C \times D \times D$, where N is the number of examples in training, C is the number of channels, and D is the target size of the inputs and outputs. For the training of model A, $C = 3$, encompassing the velocity components (U, V) and pressure (P). For the training of model B, an additional channel was introduced to represent geometric mask domains. To better illustrate the application of warm-up data in our training process, Table 1 summarizes the input datasets and their corresponding sizes for different models. This table facilitates

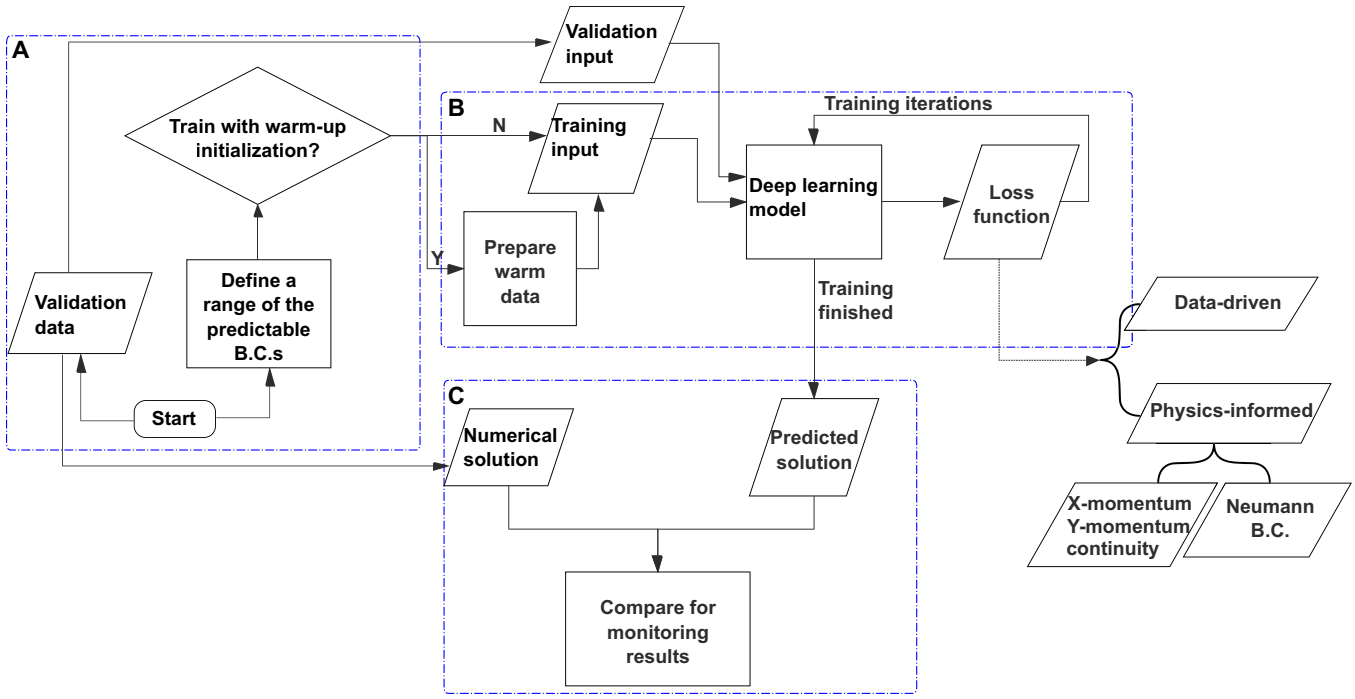

Fig. 2. Workflow of the proposed method. (A) For each training experiment, predictable boundary conditions (B.C.s) are first defined, and training and validation data are prepared for training and validation purposes separately. The training data are prepared for the deep learning model with either plain initialization or warm-up initialization dependent on the problems. (B) Training is performed with the given data and guided by the pre-defined loss functions, which consist of a hybrid of data-driven and physics-driven loss. (C) Validation data are prepared to give the testing result the proper initial state. A predicted solution corresponding to the given input is generated by a trained model.

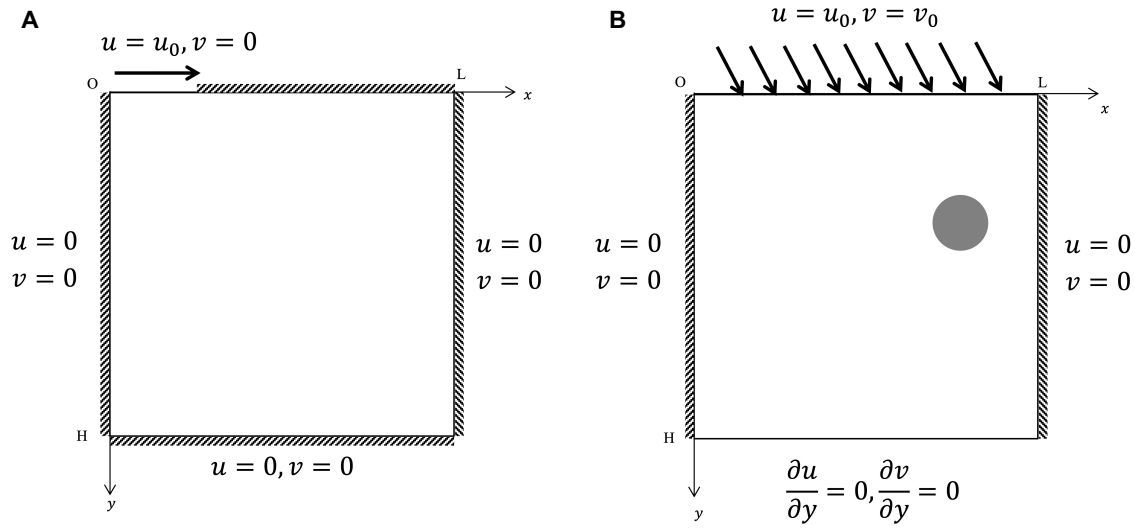

Fig. 3. Schematics of the example problems. (A) Cavity flow problem with moving lid with different lid velocities U_0. The moving part of the cavity lid can be changed for model A. (B) Inclined flow passing over obstacles at different inlet velocities (U_0, V_0). The obstacles can have different sizes and shapes. Pressure $p = 0$ is applied at the origin of the square domain.

a clear understanding of the variations in input data across the stages of model development, highlighting the transition from warm-up data to cold input as the models progressed.

Pre-run solutions

For the lid-driven cavity flow problem, we utilized pre-run solutions to generate a warm-up dataset of 2,048 samples. These samples were designed to cover a range of lid velocities, uniformly distributed from 0 to 0.5, corresponding to Reynolds numbers (*Re*) from 0 to 10, with viscosity held constant across scenarios. This dataset has a size of $2{,}048 \times 3 \times 32 \times 32$ to acclimate the model to a variety of flow conditions, enabling it to learn fundamental flow dynamics before being exposed to more nuanced variations.

Coarse solutions

For scenarios involving flow around obstacles, we used coarse solutions to create another 2,048-sample dataset. These samples were based on computational solutions for internal flow problems without obstacles, calculated on a coarse 8×8 domain. Input

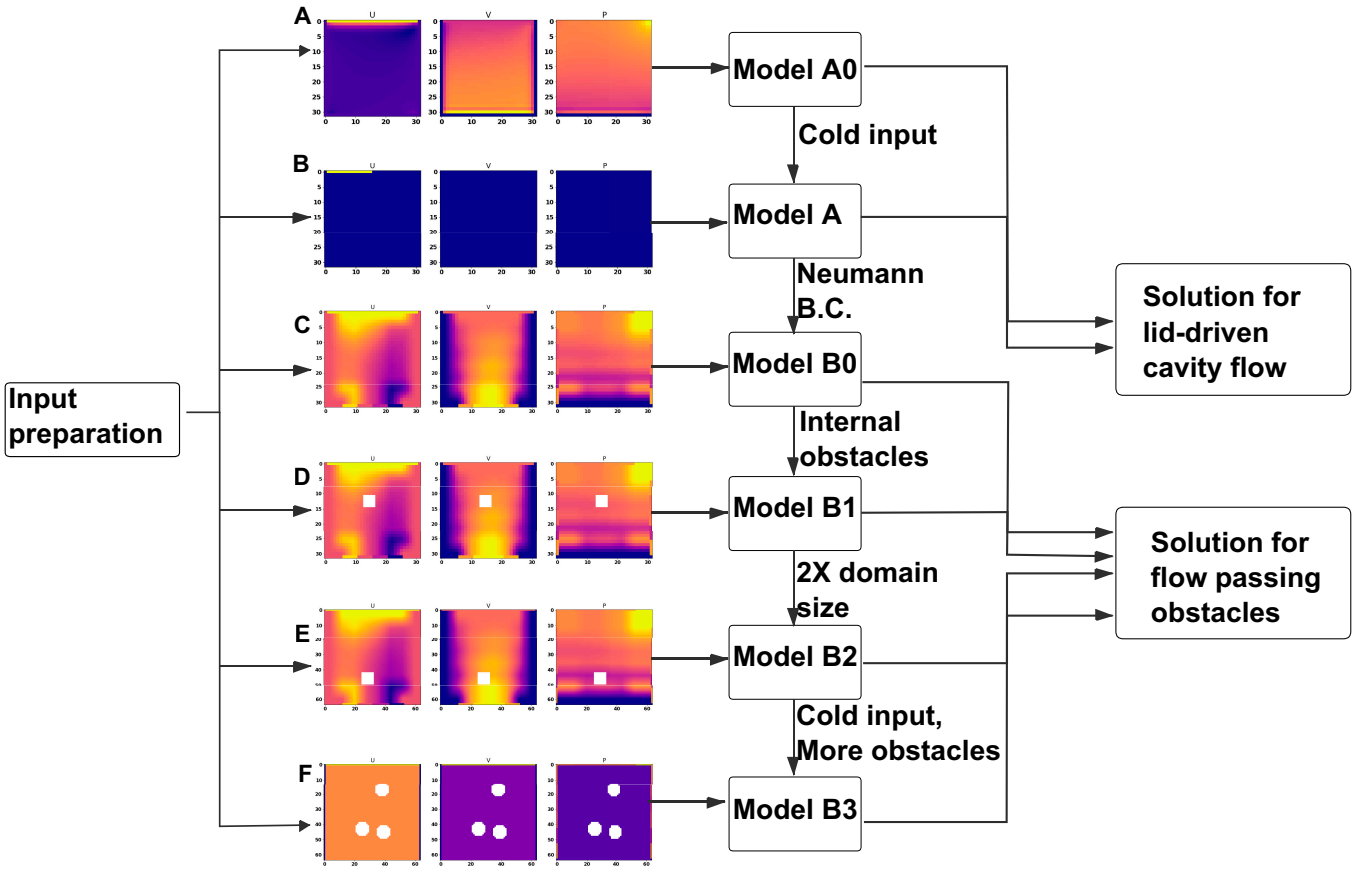

Fig. 4. Demonstration of progressive training steps for the stacked models. In the beginning, the model starts from scratch and takes warm-up initializations as the input. To reach a model with a higher level of capability, the training is split and conducted progressively, with each step validating the corresponding result. The important new capability for the upgraded models is labeled correspondingly. (A) Model A0 takes input produced by pre-run iterations. (B) Model A is trained to take input with only the boundary conditions. (C and D) B0 and B1 are the models that use coarse solutions and warm data with a single obstacle. (E and F) B2 and B3 are the models that handle larger domains. Model B3 can be trained to take only boundary conditions and obstacle geometry as input information.

Table 1. Warm-up data input for training different models

Model	Input dataset	Input data size
A0	Pre-run solutions	(2,048, 3, 32, 32)
A	Cold input	(2,048, 3, 32, 32)
B0	Coarse solutions	(2,048, 3, 32, 32)
B1	Coarse solutions	(2,048, 4, 64, 64)
B2	Coarse solutions	(2,048, 4, 64, 64)
B3	Cold input	(2,048, 4, 512, 512)

velocities, both horizontal and vertical, were uniformly sampled from a range of 0 to 0.5 at the inlet of the domain. We then interpolated these coarse solutions to the targeted resolutions to match the intended input size for the models so that the dataset had a size of $2{,}048 \times 3 \times 32 \times 32$ for training model B0 and larger domain sizes for models B1, B2, and B3. For the series of model B experiments, which introduced obstacles, a geometric mask was added as an extra input channel. This mask, indicating the positions and dimensions of obstacles, did not alter the computed results for the other 3 channels. Instead, it only provided spatial context to the warm-up dataset, enabling the model to more accurately predict flows in environments with obstructions.

Training configuration

Our models were trained for 2,000 epochs on an NVIDIA 2080 Ti GPU (graphics processing unit). The model was optimized using the Adam optimizer with a learning rate of 2×10^{-5} for models A0 and B0 and a learning rate of 1×10^{-5} for models A, B1, B2, and B3. We conducted manual tuning of hyperparameters, including the multiplier of each loss, λ_N, λ_1, λ_2, λ_3, λ_b. Experiments related to the multiplier of each contribution of loss were required so that a converging loss could be reached. The multipliers should balance the contribution of each term of loss so that each of the subterms can reach the same magnitude.

Results and Discussion

Base model for solving lid-driven cavity flow

We initially trained a model to generate a solution to the lid-driven cavity flow problem. Figure 3 shows the enclosure, a square with a lid that moves from left to right, and the boundary conditions. The computational grid is discretized by 32×32 cells.

Without a pre-trained model, the model was initialized from a *Kaiming* initialization [41]. The model was trained under weak supervision with low-cost computations as a warm-up

(Fig. 4A). We first used the pre-run warm-up dataset. It did not provide a true solution to the problem but instead simplified optimization by acting as a pre-run iteration. The only cost associated with generating the training data involved running an FDM iterative solver for 20 iterations, where the Reynolds number was defined as $Re = U_0 L/\nu$, in which L is the cavity length. The lid velocities range from 0 to 0.5, corresponding to Re from 0 to 10. Re can be changed by changing the lid velocity. After the first model was trained, the target output (U, V, P) could be generated for different Re with given pre-run steps as the input.

With the trained model A0, we conducted a series of training stages to upgrade the model for more complexity (Fig. 4). The more advanced models have more capability to produce solutions for more complex scenarios. The first upgrade is model A, which is designed to take in only "cold inputs"—zeros as the initial values. Due to the pre-trained base model A0, there was no need to run a numerical solver to sample an input for model A. Since the model was already preconditioned, the input of the second model did not require warm-up. We increased the complexity of the model by including changes to the size and position of the upper boundary. A flowchart demonstrating the training procedures is shown in Fig. 4A and B. The output from models A0 and A is shown in contour plots of the steady-state solution of the classic lid-driven cavity flow problem (Fig. 5A), and the lid-driven flow problem with movable lid location and varying sizes (Fig. 5B). The accuracy and inference speed of models A and A0 are described in Table 2.

Advanced model for solving flow over obstacles

After demonstrating the efficacy of our approach with lid-driven cavity flow, we further expanded this concept to tackle more complex problems involving inclined flow passing over obstacles at various inlet velocities. We trained a base model, again, in a weakly supervised manner, leveraging simulated data. The base model lays a foundation for a more generalizable model, capable of processing inputs without reliance on simulations. To demonstrate the method and test its extensibility, we designed more advanced models that contain the larger domain with Neumann boundary conditions and internal obstacles. Additionally, we included a physics-informed contribution to the loss of Neumann boundary conditions at the domain boundaries. In this example, we trained a model to generate solutions for laminar internal flow problems with internal obstacles. A schematic of the example is shown in Fig. 3B. We consider the boundary conditions imposed at the velocity inlet and pressure outlet. The fluid flow is inclined to enter the computational domain with inlet velocity defined by $0 \le U_0$, $V_0 \le 0.5$ so that the maximum magnitude of the velocity is $V \le \sqrt{2}/2$. The obstacles allowed in this domain do not have a characteristic length larger than the size of the domain, so the Reynolds number is small enough to be considered as laminar flow.

Following the progressive training procedures, a new base model (Fig. 4C), model B0, was developed. As in the cavity flow problem, this model was trained with the precondition of warm-up data, which uses the coarse computational results with a domain size of 8×8 interpolated to the domain 32×32 as the input of the model. This base model was trained to solve the internal flow problem. At this stage, the model input remains 3-channel. The steady-state solutions of the domain with different values of horizontal and vertical velocities can be generated by the base model.

To facilitate approximation with internal obstacles, we introduced a geometric mask, which defined the obstacles, as an additional input channel. The convolutional filter of the target model incorporates an extra channel. Weights for filters associated with the 3 channels representing the boundary are transfer-learned from the base model, while the boundary channel is optimized from scratch (Fig. 6A). We applied transfer learning in this manner to maintain the warm-up data training, even though the structure of the model changed due to the additional channel (Fig. 4D). In model B1, the shape and size of the internal obstacle are fixed, but the location of this object can vary.

To enhance predictability, model B1 was transferred to another intermediate model, model B2, by increasing the computational domain size. The depth of the fully convolutional neural network changes when the input size differs. For

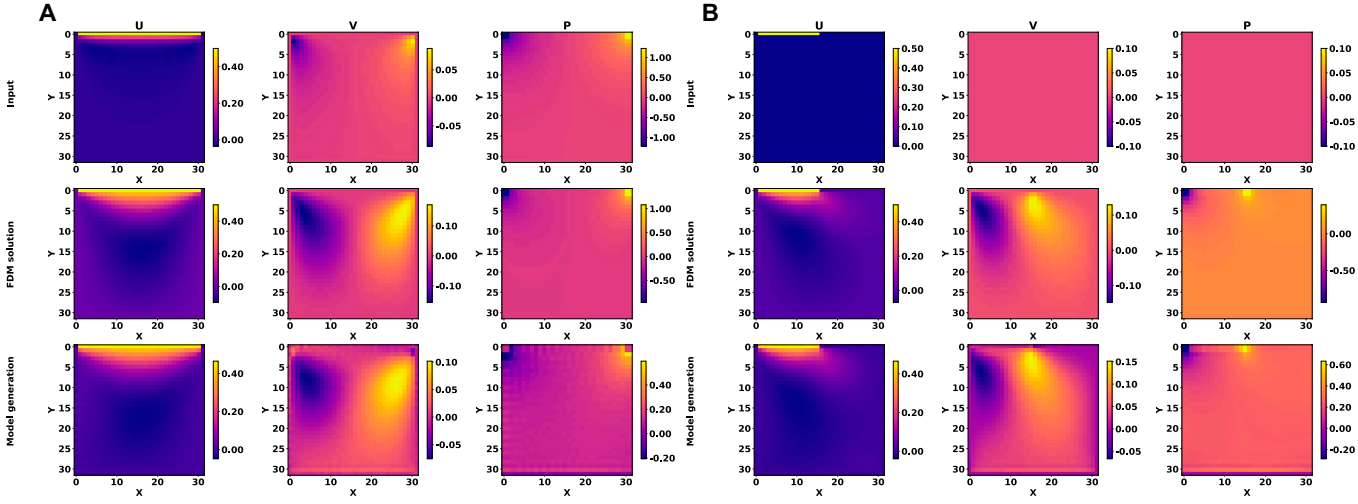

Fig. 5. Contour plots showing inputs (top), FDM solutions (middle), and model-generated solutions (bottom) for lid-driven cavity flow. (A) Solutions generated by model A0 tested on lid-driven flow. The lid velocity is $U_0 = 0.5$, corresponding to $Re = 10$. (B) Solutions generated by model A tested on modified lid-driven cavity flow. The input of the model is costless. The lid is partially moved (half-opened), and the lid velocity is $U_0 = 0.5$, corresponding to $Re = 10$.

Table 2. Summary of the results from example models

	Model			
	A0	A	B2	B3
Inlet velocities	$U=0.5$	$U=0.5$	$(U,V)=(0.05,0.5)$	$(U,V)=(0.2,0.5)$
Internal obstacles	No obstacles	No obstacles	1 square obstacle	Multiple obstacles
Warm input time (s)	0.057		0.17	
Model inference time (s)	0.0034	0.0034	0.0047	0.0064
RMSE velocity U	0.0381	0.0196	0.0186	0.0836
RMSE velocity V	0.0362	0.0438	0.0313	0.0766
RMSE pressure P	0.1418	0.2437	0.0619	0.2908

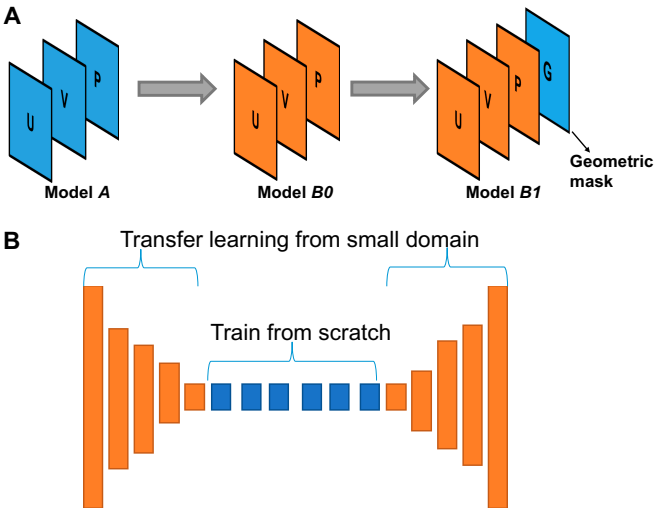

Fig. 6. Demonstration of transfer learning in training when the model has a structure change. (A) Model B0 is developed from model A, the weights of the model are transferred for training model B0, and a geometric mask is added when training model B1, with the base model transfer learning from B0. (B) During the step to improve the model from taking the input size of 32×32 to 64×64, transfer learning is applied in such a way that the inner layers are learned from scratch, while the layers at the start and the end are transferred from the previous model.

this instance, transfer learning from model B1 was performed for the layers near the beginning and end of model B2, while the inner layers necessitate training from scratch to accommodate a domain size twice as large (Fig. 6B).

Finally, we applied transfer learning, as previously outlined, to accommodate circular and square obstacles. Without any modifications to their architecture, the target models were trained and could be tested without the need for warm-up data, thereby eliminating the cost of preparing input. We upgraded the model by adding one additional layer in both encoder and decoder in this model to produce model B3, which can accept inputs with an extended domain size of 512 × 512.

From the beginning of model B0 to our final target, the only data costs incurred were from the coarse CFD simulation results of pure internal flow solutions, sampled from an obstacle-free 8 × 8 domain. The results from model B2 are shown in Fig. 7, with both the contour plots (Fig. 7A) and velocity profile plots at selected lines (Fig. 7B). Models B2 and B3 could take input of the inclined inlet velocity (U_0, V_0) within the preset range $[0, 0.5]$. Model B2 was capable of generating solutions when given any location for the fixed-size square internal obstacle. Model B3, an enhanced model, could handle multiple obstacles with different shapes and sizes on a 512 × 512 domain. The outputs of model B3 with circular and square obstacle inputs are shown in Fig. 8 (A to H) and Fig. 9 (A to H), respectively. The accuracies of both models were calculated as the RMSE and are listed in Table 2.

Discussion

We presented a highly extensible and low-cost method for solving the steady-state N-S equations under various boundary conditions and with internal obstacles. Model A0 was successfully trained to generate solutions to the steady-state N-S equations for the lid-driven cavity flow problem with the help of warm-up data, described in the Materials and Methods section, as the initialization and training dataset. Subsequently, we trained model A to solve the cavity flow problem, which takes cold input. Utilizing the simple model embedded with the N-S equations as a foundation, we trained a series of models (B0 to B3) to address different flow problems with increasing complexity, accounting for flow passing over obstacles and for larger domain sizes, ultimately providing a comprehensive solution to the steady-state N-S equation. By transfer-learning a model trained on simple problems, an improved model was trained to resolve the unseen complicated scenarios that require dealing with different boundary conditions, domain sizes, and geometric masks. Unlike typical supervised learning, in which sufficient inputs and corresponding target outputs should be prepared as labeled data, the presented models in all training experiments do not need to fit any labeled dataset that represents the whole expected physics. Therefore, these models do not rely on large CFD datasets with existing solutions. Instead, the models take advantage of low-cost pre-inputs obtained from an FDM solver running in minutes on a desktop computer.

Generating solutions at different stages requires a combination of physics-informed and boundary losses to constrain the model. At the beginning of each training experiment, the solutions generated by a model are noisy and inaccurate. The

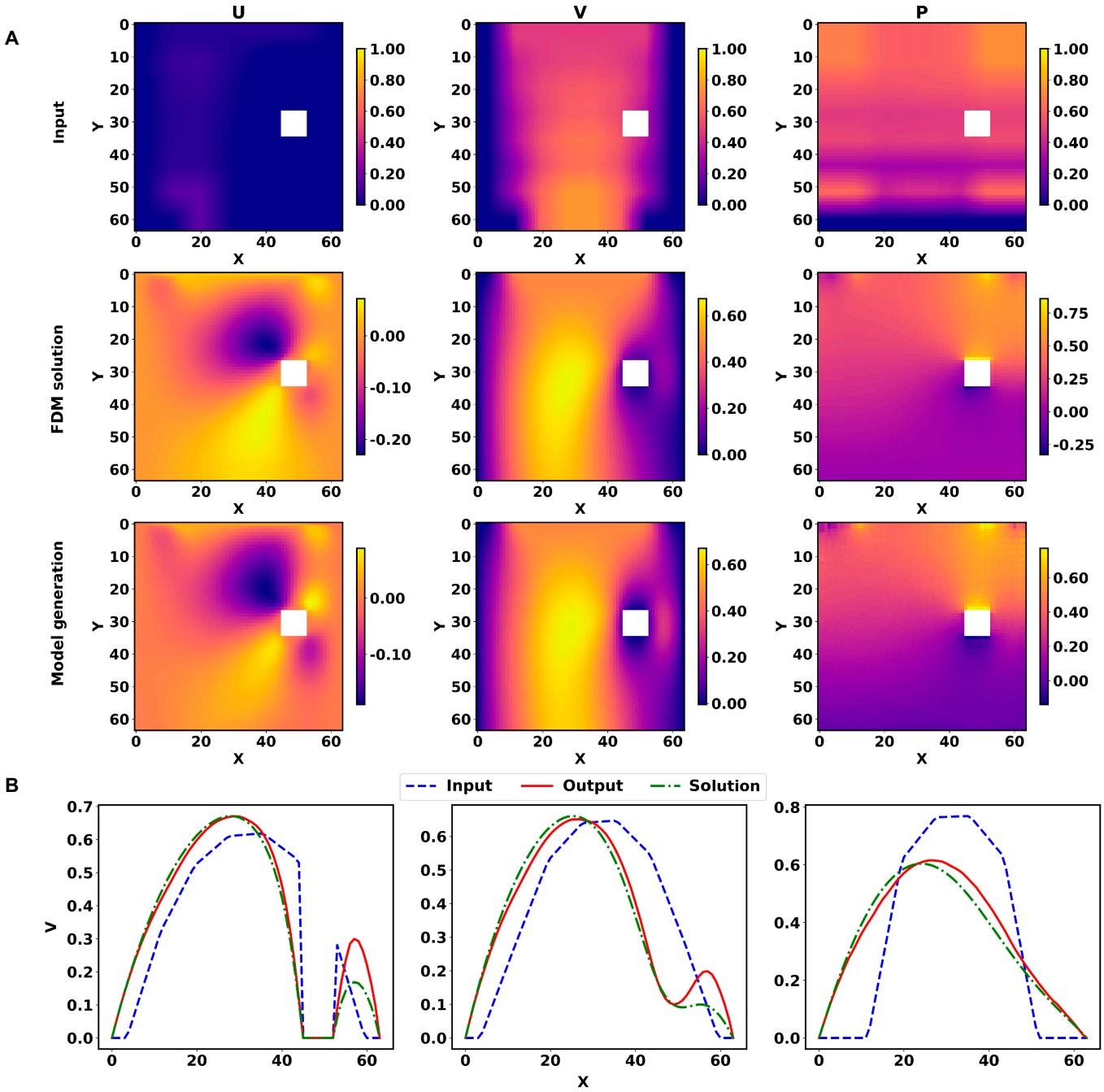

Fig. 7. Solutions generated by model B2 tested on an inclined velocity inlet and a single square obstacle with $(U_0, V_0) = (0.05, 0.5)$. (A) Contour plots. (B) Velocity profiles corresponding to the cross-section of the center line where the obstacle is located (left), $Y = 40$ (center), and the outlet (right).

total loss of the training experiments constrains the relationships of internal nodes by setting up the physics-informed loss. During the training, it is observed that if the loss solely contains the physics-informed loss, it will lose the constraint from the boundary leading to a result where the solutions generated are predominantly close to zero. The data-driven loss applied on the boundary with known boundary conditions provides the necessary constraint between the internal area and the boundary. By monitoring the contribution of each subloss, the hyperparameter for each contribution to the loss balances these constraints. The history of these 2 sublosses indicates a smooth convergence of boundary loss, while the physics-informed loss, as the substantial term, converges first rapidly and then much slower (Fig. 10).

Our method combines the strengths of traditional CFD methods and supervised learning algorithms. Without the need to convert the inputs as 1D discrete nodal points reported in PINNs, our models maintain an analogous data representation similar to that of conventional CFD approaches. Our model employs a 2D matrix structure to represent the computational domain, as in conventional CFD approaches. This format allows for a direct and intuitive input of the unknown variables, which are seamlessly integrated with specified boundary conditions and geometric constraints. By adopting this matrix representation,

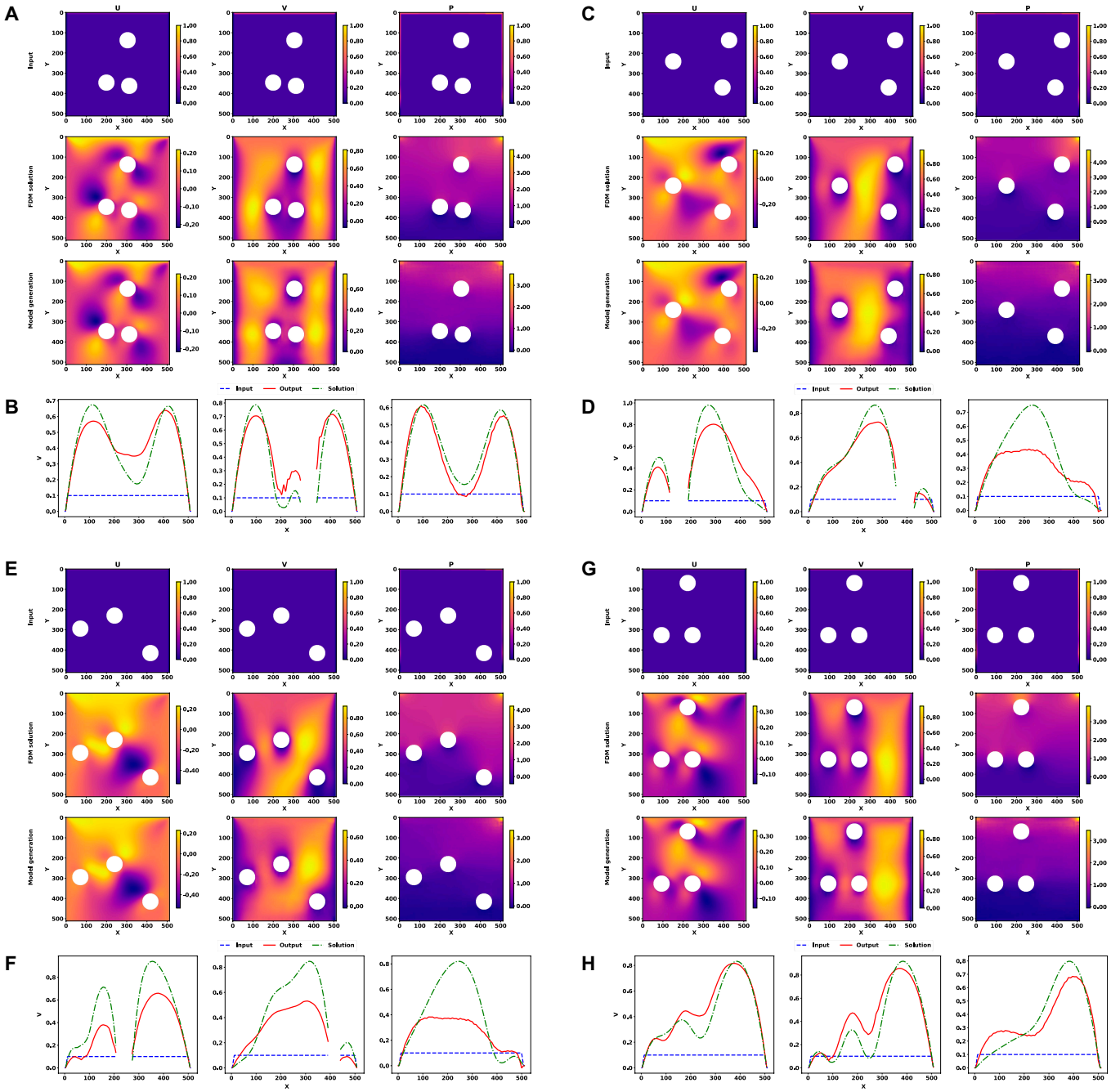

Fig. 8. Solutions generated by model B3 tested on an inclined velocity inlet and circular obstacles in different geometric configurations, with $(U_0, V_0) = (0.2, 0.5)$. (A, C, E, and G) Contour plots of the inputs and FDM and model-generated solutions. (B, D, F, and H) Velocity profiles corresponding to the cross-section of the center line $Y = 256$ (left), $Y = 384$ (center), and the outlet $Y = 512$ (right).

we facilitate an efficient embedding of essential simulation elements, including boundary conditions and any geometric features or obstacles, into the input data of the model. From training to testing, the nodal points of the input retain their structure as an image, instead of being converted to isolated discrete inputs. This representation allows the direct specification of geometric configurations and the assignment of boundary conditions. Moreover, the method is similar to a typical ML algorithm, in which the training and testing of a model are independent. During the training process, the model learns the physics, which is not restricted to a single computational configuration. Once the model is trained, it can approximate the solutions for multiple configurations of boundary conditions and geometries. The model considerably benefits from the high-speed calculation capabilities inherent in DL-based CFD simulation where the DL model can be trained offline, enabling nearly instantaneous predictions of unseen cases compared to conventional methods, as highlighted in [42]. The inference mode of the model only demands the computation of a forward pass with a fixed set of parameters. Given the highly parallelizable nature of the forward pass, the trained model generates solutions without the need for high-performance computational

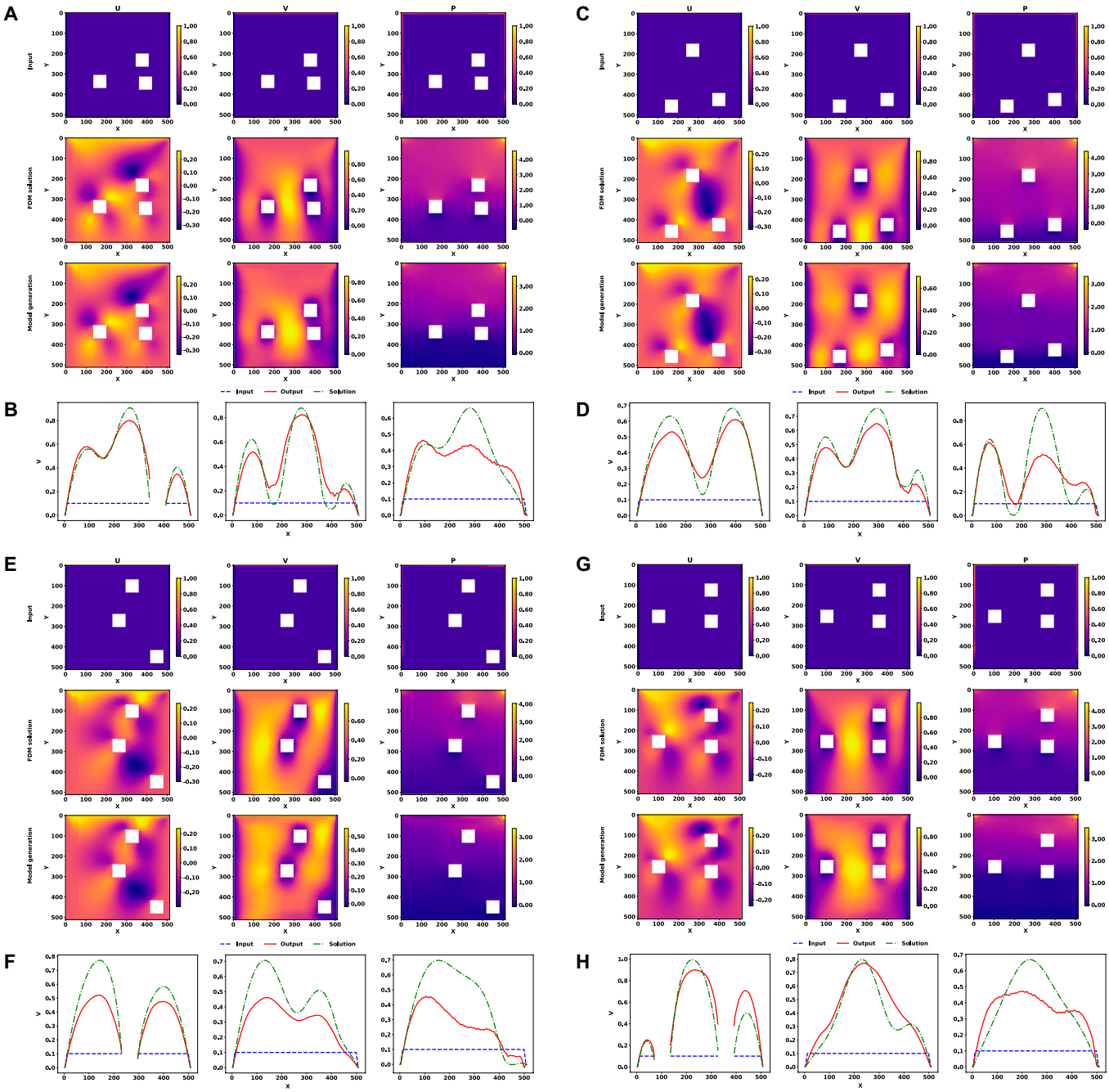

Fig. 9. Solutions generated by model B3 tested on an inclined velocity inlet and square obstacles in different geometric configurations, with $(U_0, V_0) = (0.2, 0.5)$. (A, C, E, and G) Contour plots of the inputs and FDM and model-generated solutions. (B, D, F, and H) Velocity profiles corresponding to the cross-section of the center line $Y = 256$ (left), $Y = 384$ (center), and the outlet $Y = 512$ (right).

resources. After training, the model can be deployed on lightweight computational resources. To demonstrate the speed, we benchmark the models on a consumer desktop system with an Intel Core i5 8400 processor (6 cores and 6 threads). We achieve inference latencies within 7 ms per input. This performance is in sharp contrast to the 10 s required by a corresponding FDM solver to produce a computational result on our training platform, which is equipped with NVIDIA 2080Ti and therefore possesses considerably greater computational resources. Comparable advancements in computational speed have also been documented, as evidenced by [43] reporting a speed increase of 45 times and [44] achieving an acceleration of 4,000 times, further emphasizing the considerable impact of DL-based models on CFD simulations. A visualization of the solutions instantly generated by the model can be found in the repository [45]. This accessibility enables individuals with minimal DL knowledge to generate CFD simulation results in real time, without the need for an extensive mathematical background or a CFD solver.

One of the challenges when developing and maintaining a traditional CFD solver is the necessity of running the same algorithm on different platforms with similar performance [46]. The portability of our model enables training on cloud-based GPU resources and instant solution generation on any local

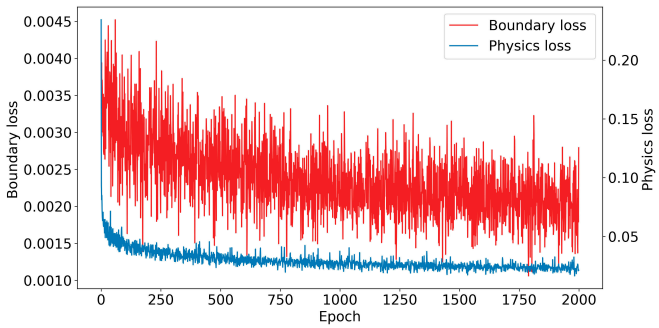

Fig. 10. Training histories for sublosses for physics and boundary losses. The 2 sublosses are not in similar scales; thus, they are plotted on a dual-axis chart.

and portable devices equipped with only basic computing functionality, thus circumventing this issue.

While our demonstration is limited to solving the 2D steady-state and laminar flow equations with homogeneous fluid properties, this concept can be extended and adapted to other problems. For example, the model can be expanded to accommodate different geometric configurations, a wider range of predictability, different physics-informed loss, and ML-based methods for other forms of flow equations such as Stokes flow [38].

Conclusion

A generative DL model implemented based on a convolutional U-Net has been developed to generate the numerical solutions for N-S equations. The trained model is capable of generating steady-state solutions for various boundary conditions and geometries in real time on consumer computing resources. The training was facilitated by warm-up initialization and does not need computational or experimental solutions as labeled training data. The training was performed in stages, adding constraints of increasing complexity, and through the inclusion of a physics-informed and data-driven loss function. Similar to traditional CFD methods, we structure the computational domain as 2D matrices, which efficiently integrate unknown variables with boundary conditions and geometric constraints into the input. The methods were validated by solving example problems including cavity flow and flow that passes obstacles. The solutions generated by a series of stacked models showed that the trained models can produce steady-state solutions to given boundary conditions for both Dirichlet and Neumann boundaries and optional internal obstacles with limited cost for training data, with good predictability, extensibility, and interpretability. This base model could be extended to solve problems with larger domains or new complexities through the addition of additional channels and subsequent optimization. Given the proper definition of the loss function, the method could also be applied to solve other PDE equations and geometric configurations. We expect that the model can be generalized to speed up general boundary-value CFD problems and, in the future, be extended to solve other fluid–structure interaction problems with minimal requirement of data.

Acknowledgments

Funding: The authors acknowledge support for this work from the National Institutes of Health (grant R01HL131750) and the National Science Foundation (grants CBET 2039310 and OAC 2215789). J.C.A. acknowledges primary support from the National Science Foundation (grant TRIPODS + X: RES-1839234).
Author contributions: S.W. developed the methods, implemented the machine learning model, performed experiments, and contributed to the writing of the manuscript. M.N. analyzed the fluid mechanics and contributed to the writing of the manuscript. J.C.A. supervised the design of the model, guided the experiments, and contributed to the writing of the manuscript. Y.L. designed the research project, analyzed results, and contributed to the writing of the manuscript.
Competing interests: The authors declare that they have no competing interests.

Data Availability

All data and source code to obtain the numerical results of the present study can be found at: https://github.com/Yaling-Liu-Lab/Generate_CFD_by_DL.

References

1. Ferziger JH, Perić M. *Computational methods for fluid dynamics.* Berlin, Germany: Springer Verlag; 1999.
2. Versteeg HK, Malalasekera W. *An introduction to computational fluid dynamics: The finite volume method.* Harlow, England, UK: Pearson Education; 2007.
3. Chen H, Chen S, Matthaeus WH. Recovery of the Navier-Stokes equations using a lattice-gas Boltzmann method. *Phys Rev A*. 1992;45(8):R5339–R5342.
4. Nikfar M, Razizadeh M, Paul R, Liu Y. Multiscale modeling of hemolysis during microfiltration. *Microfluid Nanofluidics*. 2020;24(5):33.
5. Tan J, Keller W, Sohrabi S, Yang J, Liu Y. Characterization of nanoparticle dispersion in red blood cell suspension by the lattice Boltzmann-immersed boundary method. *Nanomaterials*. 2016;6(2):30.
6. Girault V, Raviart P-A. *Finite element approximation of the Navier-Stokes equations.* Berlin, Germany: Springer; 1981.
7. Reddy JN. *Introduction to the finite element method, 4th edition.* New York, New York, USA: McGraw Hill Professional; 2018.
8. Nikfar M, Ashrafizadeh A. A coupled element-based finite-volume method for the solution of incompressible Navier-Stokes equations. *Num Heat Transf B Fundam*. 2016;69:447–472.
9. Sengupta S. *Numerical grid generation in computational fluid mechanics '88*. Swansea, Wales, UK: Pineridge;1988.
10. Duriez T, Brunton SL, Noack BR. *Machine learning control—Taming non-linear dynamics and turbulence.* Berlin, Germany: Springer; 2016.
11. Zuo W, Chen Q. Real-time or faster-than-real-time simulation of airflow in buildings. *Indoor Air*. 2009;19(1):33–44.
12. Stam J. Real-time fluid dynamics for games. Paper presented at: Proceedings of the Game Developer Conference; 2003; San Jose, CA, USA. p. 25.
13. Omerdic E, Toal D. Modelling of waves and ocean currents for real-time simulation of ocean dynamics. Paper presented at: OCEANS 2007—Europe; 2007; Aberdeen, Scotland. p. 1–6.
14. Biswas R. *Parallel computational fluid dynamics: Recent advances and future directions.* Lancaster, Pennsylvania, USA: DEStech Publications Inc.; 2010.
15. Salamonowicz Z, Majder-Lopatka M, Dmochowska A, Piechota-Polanczyk A, Polanczyk A. Numerical analysis of

smoke spreading in a medium-high building under different ventilation conditions. *Atmosphere*. 2021;12(6):705.
16. Adler PM, Jacquin CG, Quiblier JA. Flow in simulated porous media. *Int J Multiphase Flow*. 1990;16(4):691–712.
17. Zhang R, Chen Z, Chen S, Zheng J, Büyüköztürk O, Sun H. Deep long short-term memory networks for nonlinear structural seismic response prediction. *Comput Struct*. 2019;220:55–68.
18. Yang C, Yang X, Xiao X, Yang X, Xiao X. Data-driven projection method in fluid simulation: Data-driven projection method in fluid simulation. *Comput. Animat Virtual Worlds*. 2016;27(3–4):415–424.
19. Tompson J, Schlachter K, Sprechmann P, Perlin K. Accelerating eulerian fluid simulation with convolutional networks. In: *International Conference on Machine Learning*. Sydney, Australia: PMLR; 2017. p. 3424–3433.
20. 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.
21. Habibi M, Dawson STM, Arzani A. Data-driven pulsatile blood flow physics with dynamic mode decomposition. *Fluids Barriers CNS*. 2020;5(3):111.
22. Rao C, Sun H, Liu Y. Physics-informed deep learning for incompressible laminar flows. *Theor Appl Mech Lett*. 2020;10(3):207–212.
23. Qi G-J, Luo J. Small data challenges in big data era: A survey of recent progress on unsupervised and semi-supervised methods. *IEEE Trans. Pattern Anal. Mach. Intell.* 2020;44:2168–2187.
24. Lu L, Meng X, Mao Z, Karniadakis GE. DeepXDE: A deep learning library for solving differential equations. *SIAM Rev*. 2021;63:208–228.
25. Kaheman K, Kaiser E, Strom B, Nathan Kutz J, Brunton S L. Learning discrepancy models from experimental data. arXiv. 2019. https://doi.org/10.48550/arXiv.1909.08574.
26. Raissi M, Perdikaris P, Karniadakis GE. Machine learning of linear differential equations using Gaussian processes. *J Comput Phys*. 2017;348:683–693.
27. Raissi M, Wang Z, Triantafyllou MS, Karniadakis GE. Deep learning of vortex-induced vibrations. *J Fluid Mech*. 2019;861:119–137.
28. Sharma R, Farimani AB, Gomes J, Eastman P, Pande V. Weakly-supervised deep learning of heat transport via physics informed loss. arXiv. 2018. https://doi.org/10.48550/arXiv.1807.11374.
29. Sun L, Gao H, Pan S, Wang J-X. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. Comput Methods Appl Mech Eng. 2020:361;112732.
30. Lu L, Jin P, Karniadakis GE, DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. arXiv. 2019. https://doi.org/10.48550/arXiv.1910.03193.
31. Ronneberger O, Fischer P, Brox T. *U-Net: Convolutional networks for biomedical image segmentation in medical image computing and computer-assisted intervention—MICCAI 2015*. Munich, Germany: Springer International Publishing; 2015. p. 234–241.
32. Farimani AB, Gomes J, Pande VS. Deep learning the physics of transport phenomena. arXiv. 2017. https://doi.org/10.48550/arXiv.1709.02432.
33. Thuerey N, Weißenow K, Prantl L, Hu X. Deep learning methods for Reynolds- averaged Navier–Stokes simulations of airfoil flows. *AIAA J*. 2020:58;25–36.
34. Werhahn M, Xie Y, Chu M, Thuerey N. A multi-pass GAN for fluid flow super-resolution. *Proc ACM Comput Graph Interact Tech*. 2019;2:1–21.
35. Feng J, Deng J, Li Z, Sun Z, Dou H, Jia K. End-to-end Res-Unet based reconstruction algorithm for photoacoustic imaging. *Biomed Opt Express*. 2020;11(9):5321–5340.
36. Hu X, Naiel MA, Wong A, Lamm M, Fieguth P. RUNet: A robust UNet architecture for image super-resolution, Paper presented at: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW); 2019, Long Beach, CA, USA. p. 505–507.
37. Souza R, Frayne R. A hybrid frequency-domain/image-domain deep network for magnetic resonance image reconstruction. Paper presented at: 2019 32nd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI); 2019, Rio de Janeiro, Brazil. p. 257–264.
38. Takbiri-Borujeni A, Kazemi H, Nasrabadi N. A data-driven proxy to Stoke's flow in porous media. arXiv. 2019. https://doi.org/10.48550/arXiv.1905.06327.
39. Zhang H, Xu T, Li H, Zhang S, Wang X, Huang X, Metaxas D. StackGAN: Text to photo-realistic image synthesis with stacked generative adversarial networks. Paper presented at: 2017 IEEE International Conference on Computer Vision (ICCV); 2017; Venice, Italy. p. 5908–5916.
40. Zhang H, Xu T, Li H, Zhang S, Wang X, Huang X, Metaxas D. StackGAN++: Realistic image synthesis with stacked generative adversarial networks. *IEEE Trans Pattern Anal Mach Intell*. 2019;41(8):1947–1962.
41. He K, Zhang X, Ren S, Sun J. Delving deep into rectifiers: Surpassing human- level performance on imagenet classification. Paper presented at: 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile.p. 1026–1034.
42. Van Quang T, Doan DT, Phuong NL, Yun GY. Data-driven prediction of indoor airflow distribution in naturally ventilated residential buildings using combined CFD simulation and machine learning (ML) approach. *J Build Phys*. 2024;47(4):439–471.
43. Xiao Y, Hotta A, Fuji T, Kikuzato N, Hotta K. Urban scale 3 dimensional CFD approximation based on deep learning: A quick air flow prediction for volume study in architecture early design stage. *CAADRIA Proc*. 2022.
44. Pajaziti E, Montalt-Tordera J, Capelli C, Sivera R, Sauvage E, Quail M, Schievano S, Muthurangu V. Shape-driven deep neural networks for fast acquisition of aortic 3D pressure and velocity flow fields. *PLOS Comput Biol*. 2023;19:Article e1011055.
45. Wang S. Generate CFD by DL. GitHub repository. 2021. https://github.com/Yaling-Liu-Lab/Generate_ CFD_by_DL.
46. Reguly IZ, Mudalige GR. Productivity, performance, and portability for compu- tational fluid dynamics applications. *Comput Fluids*. 2020;199:Article 104425.