

A generalized framework for unsupervised learning and data recovery in computational fluid dynamics using discretized loss functions


Cite as: Phys. Fluids **34**, 077111 (2022); <https://doi.org/10.1063/5.0097480>

Submitted: 28 April 2022 • Accepted: 23 June 2022 • Accepted Manuscript Online: 25 June 2022 • Published Online: 08 July 2022

 Deepinder Jot Singh Aulakh,  Steven B. Beale and Jon G. Pharoah

COLLECTIONS

Paper published as part of the special topic on [Artificial Intelligence in Fluid Mechanics](#)

 This paper was selected as Featured



View Online



Export Citation



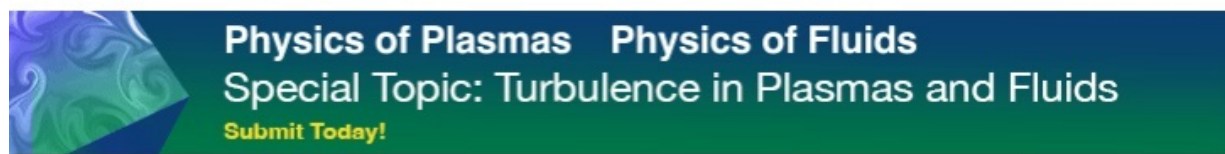
CrossMark

ARTICLES YOU MAY BE INTERESTED IN

[Physics-informed neural networks for solving Reynolds-averaged Navier–Stokes equations](#)
Phys. Fluids **34**, 075117 (2022); <https://doi.org/10.1063/5.0095270>

[Orthogonal grid physics-informed neural networks: A neural network-based simulation tool for advection–diffusion–reaction problems](#)
Phys. Fluids **34**, 077108 (2022); <https://doi.org/10.1063/5.0095536>

[Multi-scale rotation-equivariant graph neural networks for unsteady Eulerian fluid dynamics](#)
Phys. Fluids **34**, 087110 (2022); <https://doi.org/10.1063/5.0097679>



A generalized framework for unsupervised learning and data recovery in computational fluid dynamics using discretized loss functions

Cite as: Phys. Fluids **34**, 077111 (2022); doi: [10.1063/5.0097480](https://doi.org/10.1063/5.0097480)

Submitted: 28 April 2022 · Accepted: 23 June 2022 ·

Published Online: 8 July 2022



View Online



Export Citation



CrossMark

Deepinder Jot Singh Aulakh,^{1,a)}  Steven B. Beale,^{1,2}  and Jon C. Pharoah¹

AFFILIATIONS

¹Department of Mechanical and Materials Engineering, Queen's University, Kingston, Canada

²Institute of Energy and Climate Research, IEK-13, Forschungszentrum Jülich GmbH, Germany

Note: This paper is part of the special topic, Artificial Intelligence in Fluid Mechanics.

^{a)}Author to whom correspondence should be addressed: 16djsa@queensu.ca

ABSTRACT

The authors present generalized finite-volume-based discretized loss functions integrated into pressure-linked algorithms for physics-based unsupervised training of neural networks (NNs). In contrast to automatic differentiation-based counterparts, discretized loss functions leverage well-developed numerical schemes of computational fluid dynamics (CFD) for tailoring NN training specific to the flow problems. For validation, neural network-based solvers (NN solvers) are trained by posing equations such as the Poisson equation, energy equation, and Spalart–Allmaras model as loss functions. The predictions from the trained NNs agree well with the solutions from CFD solvers while also providing solution time speed-ups of up to seven times. Another application of unsupervised learning is the novel hybrid loss functions presented in this study. Hybrid learning combines the information from sparse or partial observations with a physics-based loss to train the NNs accurately and provides training speed-ups of up to five times compared with a fully unsupervised method. Also, to properly utilize the potential of discretized loss functions, they are formulated in a machine learning (ML) framework (TensorFlow) integrated with a CFD solver (OpenFOAM). The ML-CFD framework created here infuses versatility into the training by giving loss functions access to the different numerical schemes of the OpenFOAM. In addition, this integration allows for offloading the CFD programming to OpenFOAM, circumventing bottlenecks from manually coding new flow conditions in a solely ML-based framework like TensorFlow.

Published under an exclusive license by AIP Publishing. <https://doi.org/10.1063/5.0097480>

I. INTRODUCTION

The field of computational fluid dynamics (CFD) has witnessed increased applications of machine learning (ML) techniques, such as physics informed neural networks (PINNs),^{1,2} deep reinforcement learning (DRL),^{3–5} reduced-order modeling (ROM),^{6–9} turbulence closure,^{10–12} fluid–structure interaction models,^{13,14} and shape/design optimization.^{15–17} The subject of physics informed learning using PINNs has received considerable attention. PINNs are neural networks (NNs) that embed the governing equations, such as partial differential equations (PDEs), as a component of the NN itself. PINNs are trained by posing the training as an optimization problem for reducing the residual of the governing PDEs.^{1,2} The governing equations are formulated as loss functions by using automatic differentiation (AD). PINNs enhance the available data content by patching it with physics from the governing equations.¹ Hence, PINNs can be successfully applied when obtaining a large amount of training data is inhibitive.

From the point of view of CFD, physics-based learning involves training where the output of an NN is optimized to follow governing equations such as the Navier–Stokes, energy equations, and Reynolds-averaged Navier–Stokes (RANS) turbulence closure. Raissi *et al.*¹ presented pioneering work in the field of PINNs, where for loss formulations, partial derivatives of governing PDEs evaluated by using automatic differentiation¹⁸ are used to calculate residuals for backpropagating to a training algorithm. Raissi *et al.*¹ utilized PINNs for solving PDEs, such as Burger's and Allen Cahn's equations, in a non-discretized domain.

A few studies explored the concept of decomposing a solution domain into several subdomains.^{19–23} For large computational domains, Dwivedi *et al.*¹⁹ proposed distributed PINNs (DPINNs) to solve the issues of PINN robustness and vanishing gradients. DPINNs were utilized to resolve flow fields for a lid-driven flow cavity at low Reynolds numbers (*Re*). Jagtap *et al.*²⁰ addressed the issue of flux conservation between sub-domains by conservative physics informed

neural networks (cPINNs). In cPINNs, flux continuity is enforced along with sub-domain interfaces. Machine learning has also been utilized for uncertainty quantification and solution of PDEs.^{24–33}

Sun *et al.*²⁸ presented a PINN surrogate for the low Re solution of the Navier–Stokes equation on a non-discretized domain. The physics-based loss was the weighted sum of momentum and continuity equations. Furthermore, the comparison of data-driven and data-free learning approaches showed an increase in training times for data-free learning. However, data-free learning was beneficial in terms of circumventing the computational effort required for generating training data for supervised learning. Zhu *et al.*²⁹ presented a data-free surrogate for reduced-order modeling using an encoder–decoder NN for the solution of PDEs. The physics-based loss was posed as minimization of Kullback–Leibler (KL) divergence in the governing equations. This model gave better generalization on out-of-distribution test inputs compared with data-based learning.

Rao *et al.*³⁴ utilized stream functions in place of velocity to ensure divergence-free conditions commonly observed during the physics-based solution of the Navier–Stokes equations. The methodology was applied on a non-discretized domain for solving a low Re flow over a cylinder. Hsieh *et al.*³⁵ proposed NNs that learned the modifications of the iterative solver at each iteration. Here, the Poisson equation was solved, and twofold to threefold speed-ups were observed as compared to the conventional solvers while preserving a similar level of accuracy.

A. Scope

In data science applications, obtaining error-free data in a meticulous form is challenging, and available data can incur significant manual effort for its conditioning. In addition, the problem of overfitting is quite substantial in data-driven learning and requires additional regularization techniques incurring more computational effort. By contrast, the inherent regularization provided by physics-based loss functions eliminates the need for any regularization.³⁶

It is also evident from the literature described above that the physics-based approach is inherently advantageous for scientific computing as: (i) it circumvents the prohibitive cost of generating training data.²⁸ (ii) The trained NNs provide more generalization as compared to data-driven counterparts.²⁹ The advantages above motivate this study to explore data-free training focusing on CFD applications.

In parallel, the literature concerning physics-based NN training predominantly uses automatic differentiation (AD) to formulate the loss functions for backpropagating the feedback. The stiffness of computed gradients in AD may affect the accuracy, stability, and convergence of the NN training, rendering a higher number of training epochs.³⁶ In CFD, high stiffness problems are commonly encountered due to strongly coupled PDEs or flow where convective components are dominant. These problems are well resolved in CFD solvers through advanced numerical/discretization schemes and segregated solution techniques such as pressure-linked algorithms.

The present work combines the advantages of both data-free learning and discretization to propose a framework that can enable data-free training of NNs using CFD principles. Specifically, the current study is unique because it utilizes discretization schemes instead of AD to formulate and integrate loss functions in CFD algorithms. The discretized loss functions can be tempered for flow-specific discretization by choosing from various well-developed discretization schemes of CFD. Furthermore, the presented loss functions also have

modular nature; that is, any given governing equation can be posed as a loss function to integrate NN training into CFD algorithms. Leveraging this modularity, the NNs are trained to predict specific solution variables, such as Poisson's equation as a loss function for predicting pressure, energy equation for temperature, and turbulence closure models for eddy viscosity.

B. Objectives and contributions

The main goal of the present study is to use finite volume-based discretization techniques to formulate and integrate grid-based loss functions into CFD algorithms. The proposed loss functions realize the NN training by integrating with CFD algorithms like any CFD solver (PBiCG, GAMG, etc.). The CFD algorithms presented are modified to accommodate the loss functions, feedback loops, and NNs. For this study, the “semi-implicit method for pressure-linked equations (SIMPLE)”³⁷ is used for mounting ML algorithms. The mounting of NNs onto the SIMPLE algorithm enhances training generalization by allowing different multiphysics phenomena to be easily added to NN training.

The use of finite-volume methods is motivated by their inherently conservative nature and the availability of various discretization techniques for problem-specific tailoring of loss functions. Another benefit of the discretized loss functions is their compatibility with grid-based CFD packages such as OpenFOAM.³⁸ The authors utilize this compatibility to present an integrated ML-CFD framework that allows for data-free training of NNs through a two-way information exchange between TensorFlow and OpenFOAM. The integration of ML-CFD packages allows for offloading the CFD programming on an already established CFD package (OpenFOAM), circumventing the commonly observed bottlenecks by coding CFD algorithms, geometry, and equations on a standalone ML package TensorFlow.

The authors also present a novel hybrid learning methodology for combining data-driven and data-free learning for training the NNs from partially available or sparse data. In traditional supervised learning, the partially available data are not able to train NNs accurately. Instead, the hybrid loss function here extracts the information from available data and patches it with a physics-based loss function to accurately train the NNs. The presented loss functions are then validated with the SIMPLE algorithm. Here, one or multiple steps of SIMPLE are replaced with an NN solver to test the coupling of NNs with the rest of the algorithm running in a conventional CFD manner. Also, the presented NN algorithms are deployed online,³⁹ as this helps the training of NNs to be independent of any storage bottleneck(s).

II. METHODOLOGY

In the context of CFD, data-free or unsupervised learning can be regarded as a training paradigm where an NN learns by using governing equations such as the Navier–Stokes equations as loss functions. For any physics-based ML approach, loss functions play a pivotal role in constraining the NN learning to satisfy the physics involved. Hence, it becomes increasingly important to accurately pose the governing equations as loss functions. The finite-volume method is widely utilized for solving CFD problems owing to its inherent conservative nature. The proven utility in CFD makes a finite-volume formulation a natural choice for discretizing the loss functions.⁴⁰ Furthermore, by using the finite-volume method, loss functions can be tailored in a problem-specific manner similar to CFD; that is, the broad knowledge

base of CFD can be used to choose the targeted discretization techniques suited for specific problems.

Briefly, the general transport equation for a scalar property, ϕ , in integral form, for a finite-volume, V_p , centered at p , is given by Eq. (1) as follows:

$$\int_t^{t+\Delta t} \left[\int_{V_p} \frac{\partial \rho \phi}{\partial t} dV + \int_{V_p} \nabla \cdot (\rho \mathbf{u} \phi) dV - \int_{V_p} \nabla \cdot (\rho \Gamma_\phi \nabla \phi) dV \right] dt = \int_t^{t+\Delta t} \left(\int_{V_p} S_\phi(\phi) dV \right) dt, \quad (1)$$

where ρ is the density, \mathbf{u} is the velocity vector, Γ_ϕ is the diffusivity, and S_ϕ is the source term. For steady state, the first term on the left-hand side of Eq. (1) is zero. The volume integrals in Eq. (1) are converted into surface integrals through Gauss's divergence theorem to a second-order accurate discretized form [Eq. (2)]

$$\int_V (\nabla \cdot \mathbf{c}) dV = \sum_f S_f \cdot \mathbf{c}_f, \quad (2)$$

where S_f is an outward pointing area vector for the cell face, \mathbf{c} is a general vector quantity, and the subscript f signifies the value of quantity at the center of the cell face. The discretized form of Eq. (1) (steady state) is given as

$$\sum_f S_f \cdot (\rho U)_f - \sum_f (\rho \Gamma_\phi)_f S_f \cdot (\nabla \phi)_f = S_u V + S_p V_p \phi_p, \quad (3)$$

where S_u and S_p linearize the source term as $S_\phi(\phi) = S_u + S_p \phi$. Finally, a linear algebraic Eq. (4) is formed for each finite volume

$$a_p \phi_p + \sum_n a_n \phi_n = b_p, \quad (4)$$

where subscripts p and n signify the values corresponding to cell centered at p and its n neighbors, respectively. Additionally, a_p and a_n are linear coefficients, and b_p on the right-hand side contains discretized

source and spatial terms. Equation (4) couples the value of solution variable ϕ_p with neighboring ϕ_n . The solution is said to converge after a given number of iterations, if the ϕ_p -solution satisfies Eq. (4) within a certain accuracy threshold.

A. Discretized loss function

Equation (4) can be rewritten as Eq. (5) to provide the measure of deviation ($loss_p$) of any arbitrary value of ϕ_p^a from the converged solution ϕ_p at each respective time step as follows:

$$loss_p = f \left(a_p \phi_p^a + \sum_n a_n \phi_n - b_p \right), \quad (5)$$

where if ϕ_p^a is predicted by an NN, then $loss_p$ can serve as feedback for the learning algorithm. The $loss_p$ can then be backpropagated for training the NN to obtain a converged solution for a time step, t , at cell center p .

Equation (5) provides feedback for one spatial point if feedback is generated for the entire domain as shown in Fig. 1. The output of the loss function is a column vector with error/residual (mean absolute error) information for each node of the domain coupled with neighbors. The array of loss values helps with focused training. The loss function identifies the areas in the domain that are diverging from the desired solution and feedback the training loop accordingly.

Figure 1 shows the schematic of an NN being trained by the discretized loss function. The input to the NN can comprise of but is not limited to boundary conditions and spatial coordinates. As shown in Fig. 1 predicted output field of N finite-volume centers ($\phi_1^a, \phi_2^a, \dots, \phi_p^a, \dots, \phi_N^a$) is fed to discretized loss function $f(\phi)$, and the array of the residuals obtained is used as feedback to train the NN. The feedback array is of equal size to the output from the NN. Hence, each output node of the NN receives a scalar loss value corresponding to the node's predicted output scalar. For example, if the output node has ten neurons, the loss array will also be of size ten. This process is similar to training convolutional neural networks (CNNs). The feedback loss array has the size of the last layer of the CNN. Each node of

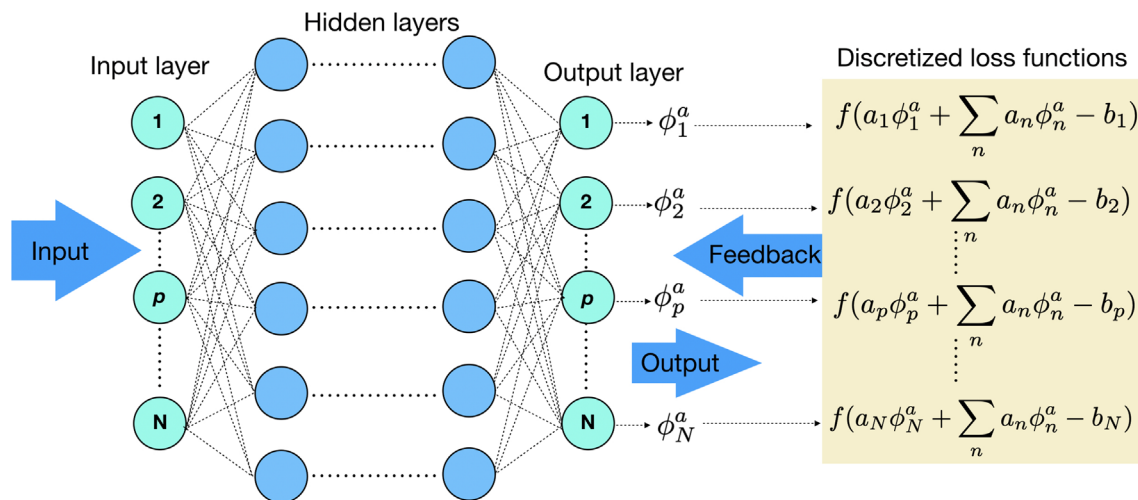


FIG. 1. Schematic of the neural network being trained by data-free discretized loss functions (yellow box). ϕ_p^a is the predicted variable at cell center p . The neural network shown here is fully connected; however, any neural network architecture can be used for the given methodology.

the CNN's last layer receives the feedback as a corresponding scalar value, which is part of a larger loss array. In this way, the NN learns to predict the correct field at every time step. If done for multiple cases simultaneously, such as different inlet velocities for flow through a channel, a batch of loss arrays can be created for training the NN. This training is similar to batch-based training in supervised learning, where multiple labeled data points are fed to a feedback algorithm for adjusting the weights of the NNs. The trained NN can be used to predict the solution field for boundary conditions not seen during training. Further details about implementation of physics-based loss function are outlined in [Appendix B 1](#).

From [Fig. 1](#), it can be seen that the NN considers full-field data in a single pass, and its size scales with the mesh size (N). Considering full-field data is a commonly used approach in the literature and is generally robust to global flow parameters.^{41,42} The NNs spanning full field are better at learning multi-scale phenomena. However, for large values of N , the NNs can have problems in scalability. The NN should be both structurally efficient (lesser trainable parameters) and independent of the mesh size to avoid scalability issues.

The convolutional neural networks (CNNs) are regarded as structurally efficient as compared to fully connected NNs. The CNNs with patch-based learning¹² can reduce the size dependence of NNs on mesh size. Also, the specialized NNs such as unstructured convolutional neural networks (UCNNs)⁴² and graph convolutional neural networks (GCNNs,^{43,44} fewer parameters than U-nets) can be explored owing to their superior performance for unstructured and body fitted meshed, respectively.

However, the focus of the current study is to develop a physics-based learning methodology in integration with CFD algorithms; specialized techniques for optimizing the network size are considered out of scope. The motivation for using the fully connected NNs was simplistic deployment due to their structurally agnostic nature.

As opposed to CNNs, the architecture of fully connected NNs does not need to consider the assumption about inputs and outputs. Hence, making fully connected NNs ideal for this study by enabling authors to focus on exploring applications of the proposed methodology. Additional details about the architecture and size of NNs are given in [Secs. IV A, V A, and VI A 1](#).

B. Data recovery or resolution enhancement using physics-based loss functions

In supervised learning, the accuracy of the trained NN is dependent on the fidelity of the data. [Figure 2](#) demonstrates this dependency, where a trained NN deviates from the ground truth due to inconsistencies in the training data. As shown in [Fig. 2\(a\)](#), the resolution of the available data is not enough to capture the oscillations in the ground truth resulting in the learned NN giving erroneous predictions. In CFD, this corresponds to the situation where the training data provided are for a coarse grid. The trained NN is susceptible to missing the small-scale phenomenon occurring in the domain. In experiments, this corresponds to sparse probe locations, where the limit on probe numbers can result in missing intricate details about the flow being measured.

The missing data can render a trained NN to deviate from the ground truth, as shown in [Fig. 2\(b\)](#). In the context of CFD, missing data on a specific location in the solution domain can be seen as either a coarse patch of mesh or simply data lost during manual conditioning. [Figure 2\(c\)](#) shows the random patches (dotted red) of information missing in a dataset of m instances of a square domain. Also, in experiments, training data can be lost due to a faulty patch of probes resulting in either erroneous or no data assimilation.

The two issues mentioned above can render the partially available data to lose its utility as partial data cannot be used to accurately train the NN by supervised learning alone. From a general perspective, the

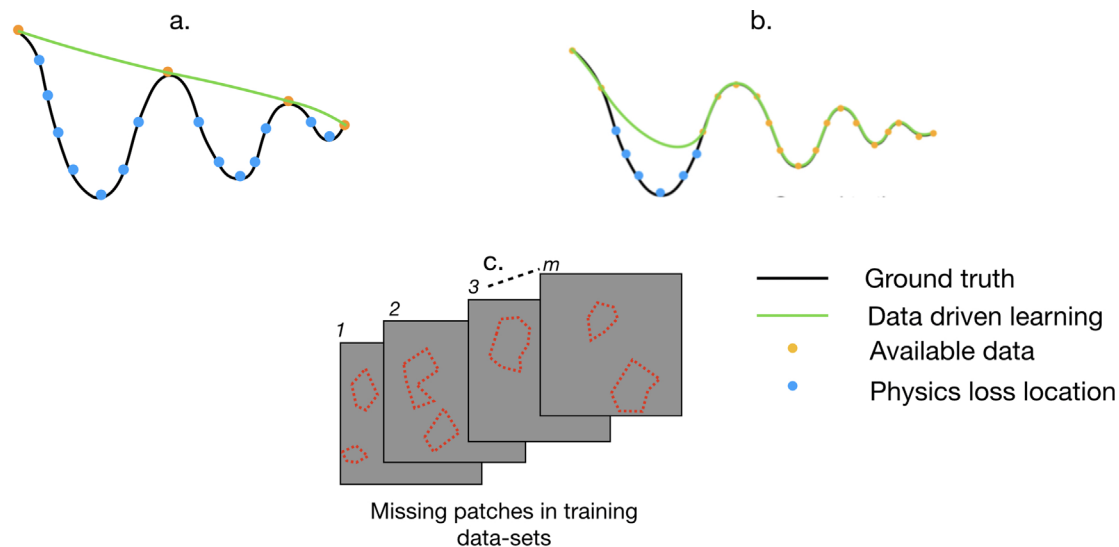


FIG. 2. Comparative example of neural network fitting with hybrid training vs supervised training. (a) Case where data available is too coarse to fully resolve the small scale oscillations. (b) Case with some parts of data missing. (c.) Shows schematic example of patches of missing data (dotted red) in a 2D domain for training dataset of m instances. These data can be missing due to malfunction of sensors in experiments, or simply a patch of mesh, which is coarse and requires further refinement to resolve flow accurately.

two issues are inherently similar; that is, for the second case, the data are missing in concentrated patches [Fig. 2(c)]. In the first case, the data can be viewed as sparsely missing throughout the domain; that is, the concentration of the available data is not fine enough to match the resolution required.

The problems above can be addressed by using *a priori* knowledge in the form of the governing equations to augment learning from the partially available data. A hybrid learning methodology is presented for using physics-based learning in conjunction with data-driven training. In hybrid learning, the loss function is a combination of the physics-based loss function and data-driven loss functions, as shown in general form by Eq. (6),

$$Loss = loss_{data\ driven} \cup loss_{physics}, \quad (6)$$

where grid points or probe locations of data-driven ($loss_{data\ driven}$) and physics-informed learning ($loss_{physics}$) are mutually exhaustive sets of the entire domain.

For the first case, the coarse data can be used as an anchor to contribute toward the data-driven loss, while for additional learning points where no ground truth data are available, a physics-based loss can be used to train the NN. Figure 2(a) shows the schematic of the first case, where the orange dots represent available data and blue dots represent the locations where the physics-based loss function is used to calculate the loss.

Similarly, for the second case with missing data, the data-driven and physics-based losses are used to train the NN where the physics-based loss function replaces the lost data, as shown in Fig. 2(b). It should be noted that another solution to the problem of resolution enhancement and recovery of missing data is to redo the entire CFD simulations (multi-grid) or experiments. However, this can be a computationally intensive process as the CFD simulation needs to solve through the domain as a whole for each instance of the training dataset. By contrast, the hybrid approach applies only to domain locations where data are missing, as shown in Eq. (7). The hybrid approach thus builds upon any previous effort invested to generate the training data, even if some portion of that data is missing. The same holds for redoing the entire experiment again,

$$Loss = g(\phi_k, \phi_k^{pred})_{data} \cup f(a_j \phi_j^a + \sum_n a_n \phi_n^a - b_j)_{physics}, \quad (7)$$

where k and j are sets of points where data are available and missing, respectively, such that $k \cup j = N$, note that N is the set of all the points in the domain. The union sign in Eqs. (6) and (7) signifies that two mutually exhaustive loss sets (data-driven and physics-based) are combined to form a unified loss array (described in Sec. II A) for the entire domain. Further details about the union are provided in Appendix B 2 and Fig. 23.

In contrast to previous studies,^{1,2,20} individual terms in Eqs. (6) and (7) are not multiplied by balancing coefficients. Apart from standard regularization, balancing/weighting coefficients are usually required when each loss term optimizes a different flow variable or equations with varying optimization characteristics. To further elaborate, in a case where one term optimizes pressure, and the other optimizes the velocity. The magnitude difference between the velocity and pressure is addressed by weighing coefficients. However, in the present study, both terms in the equation optimize the same variable; hence,

both have similar magnitudes. This similarity in magnitude eliminates the need for any balancing coefficients.

The reader will note that the problem of missing data patches is more common in experimental data. For proof of concept and brevity, in the present study, the authors only work on the CFD data missing some patches. Applying hybrid learning to reconstruct missing data for experiments is a part of more detailed future work. As the fidelity of the reconstructed data is highly dependent on the accuracy of the governing equation used to define the flow phenomena, the recovery of experimental data requires governing equations to be coupled with additional techniques such as transfer learning for accurate reconstruction.

III. ML-CFD INTEGRATION

The loss functions presented in this study are deployed using modified pressure-linked CFD algorithms. The respective NN for each solution step is trained by posing the corresponding governing equation (solved at that step) as a loss function. The present study focuses on modifying SIMPLE algorithms. However, the given methodology can be easily applied to any other CFD algorithm. The details of the modified SIMPLE algorithm are outlined in Secs. IV–VI, where NNs solve/predict solutions to equations such as the Poisson-like pressure equation, the energy equation for heat transfer, and a one-equation turbulence model. The mixed approach adopted in this study for solving some equations with NN, and the rest with a CFD solver is necessary to gain insight into coupling the dynamics of NN-based solvers with the CFD solvers. In addition, the discretized nature of loss functions facilitates the deployment of data-free learning in contemporary finite volume-based CFD frameworks such as OpenFOAM. The present study employs TensorFlow for training the NNs and OpenFOAM for defining the flow problems. A PythonC API is used to integrate OpenFOAM and TensorFlow for utilizing the functionality of both in a single consolidated platform in real-time. The PythonC API makes TensorFlow functionality available within OpenFOAM by running it in a Python virtual environment.³⁹ A bridging code takes the data from a C++ environment (OpenFOAM) and feeds it into a Python environment running TensorFlow.³⁹ The bridging code used here realizes unsupervised learning by allowing for real-time and two-way data exchange between TensorFlow and OpenFOAM. The two-way coupling between CFD and ML is also essential for stability and convergence of the NN-based solution of the governing equations.⁴⁵ In every solution iteration, OpenFOAM sends discretized loss function updated with recent field values to TensorFlow and receives the solution field predicted by NN, which is being trained simultaneously. The real-time deployment is essential for any data-free learning as NNs under training simultaneously require the residual information from the equation (used as loss function) whose solution they are predicting. The presented ML-CFD integration further advances the current state of art OpenFOAM-TensorFlow integration, which only allows for supervised learning.³⁹

The ML-CFD framework essentially provides the freedom of implementing any NN architectures and learning algorithms available in TensorFlow with any CFD algorithm or numerical schemes developed for OpenFOAM. The mutual access of internal libraries of each program to the other essentially builds a CFD-based NN trainer that can adapt to changing flow conditions like a stand-alone OpenFOAM simulation. The main benefit of this ML-CFD framework is that the task to code different geometries, CFD algorithms, and discretization schemes is offloaded to OpenFOAM. The offloaded coding effort

would have presented a significant bottleneck if a stand-alone ML application (TensorFlow) was used to code CFD also.

IV. SOLUTION OF THE PRESSURE EQUATION

In the SIMPLE algorithm, flow variables are calculated in a segregated manner, as shown in algorithm 1.^{37,46} First, the velocity field is calculated from discretized momentum equation. Non-linearity in the momentum equation occurs due to the flux term, which is itself a function of velocity computed from values at the previous iteration. The solution of the momentum equations is followed by the next step, namely, the calculation of the pressure from the Poisson equation. The pressure obtained here is used to calculate mass flux, followed by a correction of the velocity field before the start of the next iteration. For reference, a detailed description of the standalone SIMPLE algorithm is provided in [Appendix A](#).

This section uses an NN to solve the pressure equation as a loss function. Algorithm 1 shows the training and testing framework of the NN based on a fully physics-based loss function. In the training loop, the NN predicts the pressure field, which is subsequently fed to a loss formulation; the loss value thus obtained is a measure of the deviation of the solved pressure from the real solution. This loss value is then fed back to the NN for its training, in turn improving the pressure value for the next iteration of the solution. For the prediction loop, the trained NN is used to predict the pressure for given boundary conditions, and this pressure is used to calculate the velocity and flux fields for the flow problem at hand until convergence. In the prediction loop, loss calculations are not necessary.

It should be noted that in the present study, the authors are mounting NN solvers only on the SIMPLE algorithm. The motivation for using the SIMPLE algorithm stems from its use in a wide range of CFD applications. However, the current approach is valid for any other numerical algorithm.

ALGORITHM 1: SIMPLE algorithm (simpleFOAM³⁸) modified to train NNs for pressure prediction.

-
- 1: Obtain approximate velocity field (\mathbf{u}) by solving momentum equation. The pressure (P^{old}) used in this step is either from the previous iteration or an initial guess. The velocity here is not divergence free.
 - 2: **if** Conventional SIMPLE algorithm **then**
 - 3: Solve Poisson equation for pressure distribution (P^{p}). This equation is formulated using the velocity field obtained in step 1.
 - 4: **if** Training the NN **then**
 - 5: Predict the pressure (P^{p}) using NN.
 - 6: Substitute the (P^{p}) to discretized Poisson/pressure equation posed as a loss function to calculate the loss at each grid point.
 - 7: Feedback the learning algorithm for training the NN.
 - 8: **if** Predicting pressure from NN **then**
 - 9: Predict the pressure (P^{p}) from the NN. Input to the NN can be velocity field, boundary conditions or geometry.
 - 10: A new set of conservative mass fluxes is calculated using pressure P^{new} , which is obtained by under relaxing the pressure using $P^{\text{new}} = P^{\text{old}} + \alpha_p (P^{\text{p}} - P^{\text{old}})$.
 - 11: **if** Velocity is needed before next momentum solution **then**
 - 12: Correct velocities using P^{new} .
-

A. Lid-driven cavity flow

This section describes the use of an NN-based solver to resolve the flow in a lid-driven cavity. The loss function, in this case, is the pressure equation. The pressure equation effects the mass conservation by intricate coupling between the momentum and continuity equations. In the pressure equation, the complex interplay of variables such as pressure, velocity, and flux makes it ideal for testing the accuracy and coupling ability of the NN solver. The flow solved here is at Re 100. The NN solver separately uses both fully physics-based [Eq. (5)] and hybrid loss functions [Eq. (7)]. The approach of solving flow at a single Re is motivated by the work on PINNs by Jagtap *et al.*²⁰ and Jin *et al.*² Jagtap *et al.*²⁰ tested the accuracy of cPINNs using a lid-driven cavity case corresponding to Re 100. This approach is suitable for testing the accuracy of NN as a solver but does not impart the generalization to the NN to predict flow at a different Re after the solution for the training case (Re 100) is obtained. Hence, in this section, the authors did not subject NNs to the prediction loop and rather treated Poisson's equation as a proof of concept for testing the accuracy and coupling of NNs as solvers, not as predictors. However, in Secs. V and VI, the trained NNs are used as predictors for the flow problems such as turbulence closure and heat transfer.

For solving pressure by NN solver, the input to the NN is the velocity field at each iteration, and the output is the pressure field. The NN architecture consists of five levels of layers with \tanh activation. There are two input layers at the first level, each of size 1600. The x and y components of the velocity field are, respectively, fed as input to the corresponding input layer. Each input layer further feeds a corresponding fully connected layer of five neurons for the second level. The output of each five-neuron layer is concatenated at the third level and further fed to a fully connected and penultimate layer of five neurons. The production and fifth-level layer have a size of 1600 and outputs pressure field. The total number of trainable parameters here is 25 665. By comparison, Jagtap *et al.*²⁰ used 12 cPINNs for solving the lid-driven cavity flow; here, the domain was divided into 12 sub-domains corresponding to each cPINN. The present authors approximate trainable parameters for all cPINNs as 24 984 compared with 25 665 in the current study.

As discussed in Sec. II A, the present study focused on loss functions, and no optimization study was performed for NN architecture. Hence, NN scales with the mesh size. However, the size dependence of NN on the mesh and the number of trainable parameters can be reduced by order of magnitude using techniques such as patch-based learning, GCNNs.^{12,43}

1. Training hyper-parameters and comparison to PINNs

In this study, Adam optimizer⁴⁷ is used for optimizing the NN parameters. The initial learning rate at the start of the first SIMPLE iteration is 1×10^{-3} . The batch size is one since a single Re is used as a training/solution point. As mentioned earlier, choosing a batch size of one is motivated by previous studies on PINNs where a NN is used as a solver rather than a predictor. This batch size is also necessary to compare the current approach directly with the PINNs. Each SIMPLE iteration has 40 epochs for training the NN. The learning rate is scheduled both locally and globally. Globally, the initial learning rate at the start of n th SIMPLE iteration is given by $(1 \times 10^{-3})/(2 \times (n - 1))$. Locally (inside each SIMPLE loop), the global learning rate at the start

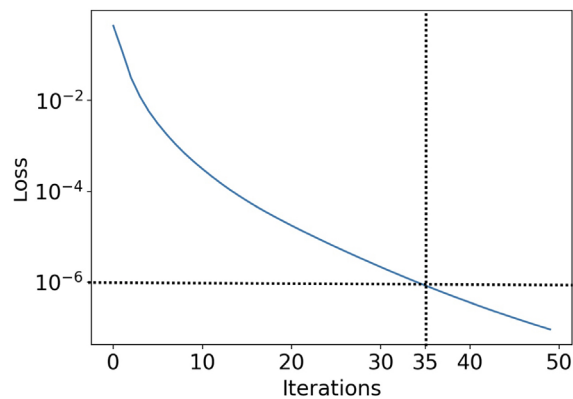


FIG. 3. Loss/mean squared error evolution for solution of pressure equation by NN solver. The loss function used here is fully physics based.

of each SIMPLE iteration is reduced by a factor of five after every five epochs.

Figure 3 shows the mean square error for a fully physics-based loss function for solving lid-driven cavity flow at Re 100. It can be seen that the loss reduced to 1×10^{-6} after 35 iterations of the SIMPLE loop. As mentioned earlier, each iteration of the SIMPLE loop has 40 epochs/back passes. Hence, in this case, the discretized loss function reached an error threshold of 1×10^{-6} in $35 \times 40 = 1400$ back passes of loss to the optimizer (epochs in supervised learning). In the study by Jagtap *et al.*,²⁰ the AD-based loss function trained cPINNs to a similar error threshold in approximately 4×10^5 epochs. Hence, the discretized loss functions are about two orders of magnitude faster than AD for similar trainable parameters (25 665 for the current study and 24 984 for Jagtap *et al.*²⁰).

In this section, the central difference scheme discretizes the loss function. The loss function, in this case, takes the form of Eq. (5) for fully physics-based and Eq. (7) for hybrid learning. Each term of Poisson's equation is linearized using finite-volume discretization as explained in the Sec. II. The linearized terms are put together by using algebraic operations such as addition and subtraction to formulate the physics-based loss in the form of Eq. (5). Here, simpleFoam (CFD solver) uses a GAMG solver for pressure and a smooth solver for velocity.

The cPINNs are chosen for comparison as they are faster to converge than standard PINNs.²⁰ The primary reason behind this speedup is that discretized loss functions take advantage of well-developed segregated solution methodology of SIMPLE to decouple the highly coupled and stiff formulation of momentum and continuity equations, whereas cPINNs try to learn this implicit coupling by simply relying on loss optimization, which is generally reliable for learning uncoupled systems but can be unstable for learning highly coupled and stiff formulations. However, the given comparison is preliminary, and more comprehensive analysis of backpropagation of error is required to understand the speedups further.

For the hybrid [Eq. (7)] loss function, a part of the pressure field missing from the available solution data in the patch is as shown in Fig. 4(c), dotted red region. Here, the authors assess the patching capability of the hybrid loss function to recover any information lost, for any given data point in a training dataset. The analysis of loss evolution in hybrid learning is given in Appendix C.

2. Validation of NN accuracy for solving pressure

Contours comparing the solutions of the CFD benchmark and the NN solver for and pressure ($P/\rho U_\infty^2$) and velocity magnitude (u/U_∞) are shown in Figs. 4 and 5, respectively. Here, u and P are velocity magnitude and pressure, respectively, and U_∞ is the velocity of the lid. It can be observed that the contour plots for the solution with the NN-based solver match well with those of the CFD solver. Also, the streamline plot [Fig. 5(b)] shows the secondary vortex in the velocity field obtained from the reconstructed pressure field.

For quantitative comparison between the two solutions, Fig. 6 provides the centerline velocity and pressure comparison for NN-based flows with CFD solver. Additionally, the results are compared to results from Ghia *et al.*⁴⁸ for further validation. It can be seen that all the results are in good agreement. Across the domain, the L2-norm error in NN solution with respect to CFD solution for pressure ($P/\rho U_\infty^2$) is 7.41×10^{-6} (fully physics based) and 8.8×10^{-6} (hybrid learning). At boundaries, the error for $P/\rho U_\infty^2$ is 7.56×10^{-6} for fully physics-based learning and 8.9×10^{-6} for hybrid learning. For velocity, (u/U_∞) the domain error is 1.96×10^{-5} for physics-based learning and 2.11×10^{-5} for hybrid learning. Hence, both loss functions

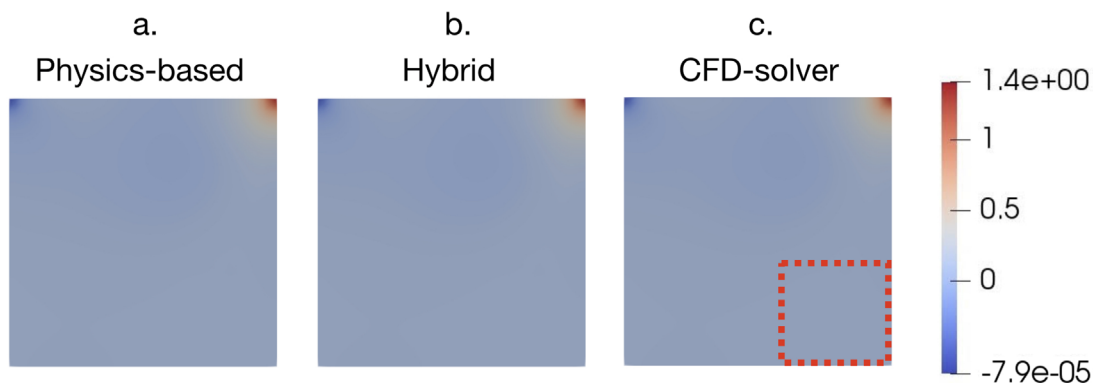


FIG. 4. Comparison of the pressure ($P/\rho U_\infty^2$) contours for the lid driven cavity problem at Re 100. (a) Physics-based loss function, (b) hybrid loss function. (c) CFD-Solver (GAMG). Also, in (c), red rectangle indicates the location where pressure data were missing and retrieved through hybrid learning.

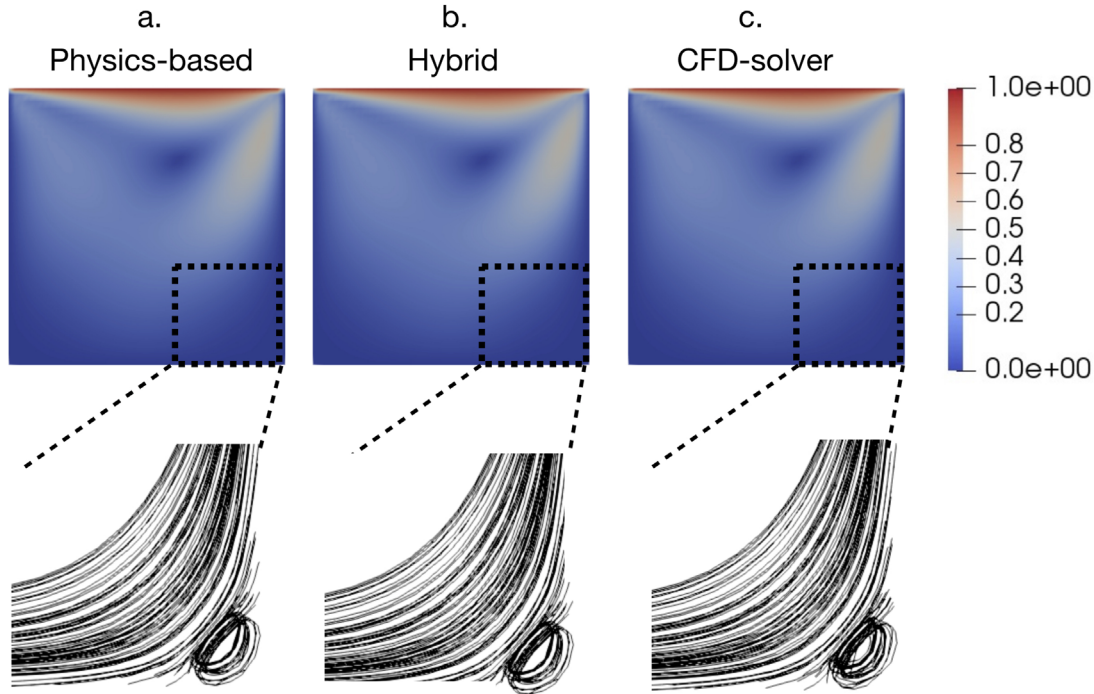


FIG. 5. Comparison between velocity (u/U_∞) contours and streamlines for the solution of lid driven cavity flow problem by CFD-based and NN-based solver at Re 100. Here, U_∞ is the lid velocity. (a) NN solver with fully physics-based loss, (b) hybrid loss function, (c) CFD solver. Additionally, the streamlines plot in (b) shows secondary vortices in the velocity patch reconstructed from recovered pressure data.

(hybrid and fully unsupervised) can give reasonably good solutions for the pressure equation.

In addition, the results show that data-driven learning augmented with physics-based loss function can be used to train the NN even with some data missing, which is not possible with a purely data-driven approach. Hence, as discussed in Sec. II B, the hybrid function helped to extract the information from the partially available data with a targeted computational effort as opposed to performing computation over the entire domain.

To reiterate, in this section the flow corresponding to only one Re for each loss function was used for training the NN; this does not provide the generalization required to predict the flow at unseen Re . The generalized NNs are trained in Secs. V and VII, where the prediction loop is run to obtain solutions at unseen flow conditions, which are not part of training solutions.

V. SPALART-ALLMARAS CLOSURE MODEL USING NEURAL NETWORKS

This section uses an NN to predict turbulent eddy viscosity for the closure of the Reynolds-averaged Navier–Stokes (RANS) equations. RANS are time-averaged equations describing the steady-state behavior of turbulent flows. In RANS, the instantaneous quantities are decomposed via Reynolds decomposition into time-averaged and fluctuating components. For an incompressible Newtonian fluid flow, the RANS equation is given by Eq. (8) as follows:

$$\rho \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} = \rho \bar{f}_i + \frac{\partial}{\partial x_j} \left[-\bar{p} \delta_{ij} + 2\mu \bar{S}_{ij} - \rho \overline{u'_i u'_j} \right], \quad (8)$$

where superscripts \bar{u} and u' denote time-averaged and fluctuating components, respectively; \bar{f}_i is an external force vector; δ_{ij} is the Kronecker delta; \bar{S}_{ij} is the mean rate of stress tensor; and $\rho \overline{u'_i u'_j}$ denotes the components of the Reynolds stress tensor. The presence of Reynolds stresses requires additional model specifications for RANS closure. One such model is the Spalart–Allmaras (SA) closure⁴⁹ for kinematic eddy turbulent viscosity ν_t . SA is a one equation model written as follows:

$$\frac{\partial \tilde{\nu}}{\partial t} + u_j \frac{\partial \tilde{\nu}}{\partial x_j} = C_{b1} [1 - f_{t2}] \tilde{S} \tilde{\nu} + \frac{1}{\sigma} \{ \nabla \cdot [(\nu + \tilde{\nu}) \nabla \tilde{\nu}] + C_{b2} |\nabla \tilde{\nu}|^2 \} - \left[C_{w1} f_w - \frac{b_1}{\kappa^2} f_{t2} \right] \left(\frac{\tilde{\nu}}{d} \right)^2 + f_{t1} \Delta U^2, \quad (9)$$

$$\nu_t = \tilde{\nu} f_{v1}, \quad (9a)$$

$$f_{v1} = \frac{\chi^3}{\chi^3 + C_{v1}^3}, \quad (9b)$$

$$\chi := \frac{\tilde{\nu}}{\nu}, \quad (9c)$$

$$\tilde{S} \equiv S + \frac{\tilde{\nu}}{\kappa^2 d^2} f_{v2}, \quad (9d)$$

$$f_{v2} = 1 - \frac{\chi}{1 + \chi f_{v1}}, \quad (9e)$$

$$f_w = g \left[\frac{1 + C_{w3}^6}{g^6 + C_{w3}^6} \right]^{1/6}, \quad (9f)$$

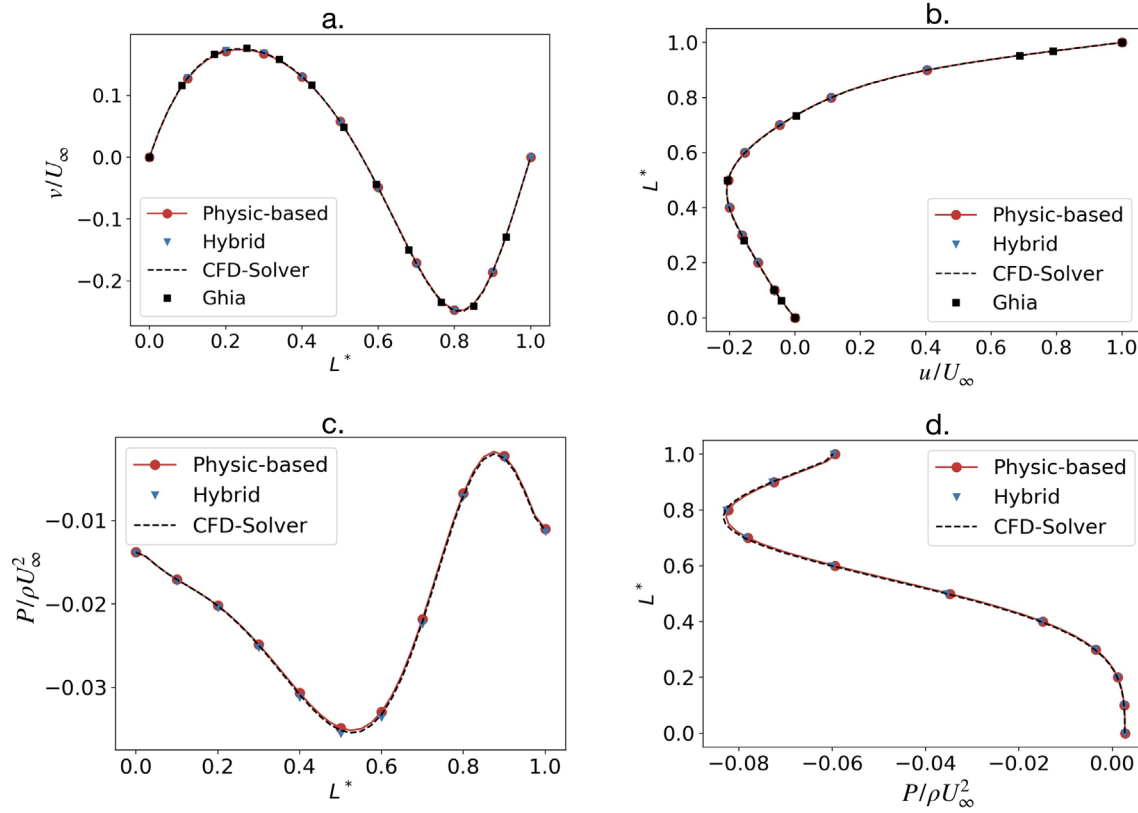


FIG. 6. Centerline plots for velocity profile at Re 100 for physics-based loss (red), hybrid loss (blue), CFD solver (dotted black), and Ghia *et al.*⁴⁸ (black squares). (a) Plots the v_y/U_∞ along the centerline parallel to x axis. (b) Compares the u_x/U_∞ along the centerline parallel to y axis. [(c) and (d)], respectively, show $P/\rho U_\infty^2$ along centerline parallel to x axis and y axis. Here, $L^* = l/L$, where L is edge length and l is the position along the edge.

$$\Omega_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i} \right), \quad (9g)$$

$$r \equiv \frac{\tilde{\nu}}{\bar{S} \kappa^2 d^2}, \quad (9h)$$

$$f_{i1} = C_{t1} g_t \exp \left(-C_{t2} \frac{\omega_t^2}{\Delta U^2} [d^2 + g_t^2 d_t^2] \right), \quad (9i)$$

$$S = \sqrt{2\Omega_{ij}\Omega_{ij}}, \quad (9j)$$

$$f_{t2} = C_{t3} \exp(-C_{t4} \chi^2), \quad (9k)$$

$$g = r + C_{w2}(r^6 - r), \quad (9l)$$

$$C_{w1} = \frac{C_{b1}}{\kappa^2} + \frac{(1 + C_{b2})}{\sigma}, \quad (9m)$$

where $\sigma = \frac{2}{3}$, $C_{b1} = 0.1355$, $C_{b2} = 0.622$, $\kappa = 0.41$, $C_{w2} = 0.3$, $C_{w3} = 2$, $C_{v1} = 7.1$, $C_{t1} = 1$, $C_{t2} = 2$, $C_{t3} = 1.1$, and $C_{t4} = 2$.

The NN in RANS solution is trained to predict kinematic turbulent eddy viscosity (ν_t) using Eqs. (5) (fully unsupervised) and (7) (hybrid). Equation (9) is discretized over the entire domain, and feedback is generated as described in Sec. II. The authors have only used the SA model, though the given methodology can also be employed to solve other turbulence closure models such as RNG $k - \varepsilon$ ⁵⁰ and $k - \omega$

SST⁵¹ as discretized loss functions. For training the NN, the loss functions are integrated into the SIMPLE algorithm as given by steps 11 to 15 of algorithm 2. The NN is invoked at the start of the SIMPLE loop for prediction. As shown in algorithm 2, steps 2–4, and the ν_t predicted by the trained NN.

In previous studies,^{33,52–55} the input to the NN is a velocity field obtained from a potential flow solution corresponding to the different boundary conditions on which the NN has to be trained. Thus, the objective of the training is to learn a relationship between the potential velocity field for a boundary condition and its corresponding steady-state ν_t . The use of potential flow as input to NN is one of the many input approaches that can be used to identify any underlying flow features that can affect the ν_t depending upon the application.

For this study, the preference of potential velocity field over actual velocity field as an input to NN stems from the problem where NN has to retrain at every iteration of the CFD algorithm, as the velocity field changes at each iteration, and so does the learned relationship between velocity field and ν_t . The NN has to learn new mapping at every SIMPLE iteration, making training computationally expensive. However, using the potential flow field (initial condition), which is constant for the entire course of the simulation, the NN needs only to refine the learning from the previous iteration for the new values of ν_t obtained corresponding to the constant potential velocity field.

ALGORITHM 2: Modified SIMPLE to train an NN for RANS turbulence closure modeling.

- 1: Calculate the potential velocity field (\mathbf{u}^p) for the given boundary conditions.
- 2: **if** Trained NN is being used to predict ν_t **then**
- 3: Predict the (ν_t) by using calculated potential velocity field (\mathbf{u}^p) as an input to the trained NN.
- 4: Repeat steps 5 to 8 until convergence.
- 5: Obtain approximate velocity field (\mathbf{u}) by solving momentum equation. The pressure (P^{old}) used in this step is either from the previous iteration or an initial guess. Here, kinematic turbulent eddy viscosity (ν_t) term is added for closure of RANS model. (ν_t) is obtained from previous iteration, initial guess or prediction from trained NN.
- 6: Solve Poisson equation for pressure distribution (P^p). This equation is formulated using the velocity field obtained in step 1.
- 7: A new set of conservative mass fluxes is calculated using pressure P^{new} which is obtained by under relaxing the pressure using $P^{\text{new}} = P^{\text{old}} + \alpha_p(P^p - P^{\text{old}})$.
- 8: Correct velocities using P^{new} .
- 9: **if** SIMPLE algorithm with conventional solver for solving ν_t for RANS **then**
- 10: Solve Spalart–Allmaras (SA) model for ν_t to be used in next iteration.
- 11: **if** NN is being trained using SA as loss function in SIMPLE algorithm **then**
- 12: Discretize the given RANS turbulence closure model to be posed as physics-based loss function. For this study it is SA model [Eq. (9)].
- 13: Predict the (ν_t) from NN, with potential velocity as input and substitute corresponding variable, i.e., $\tilde{\nu}$ into loss function to obtain the deviation at each grid point.
- 14: Feed-back the NN with the loss values for unsupervised training.
- 15: The (ν_t) obtained after reaching a certain loss threshold is fed to RANS for closure in next iteration.
- 16: Go to step 5 until convergence.

Another drawback of using the actual velocity field as input to NN is that at the end of the training, the NN learns the relationship between converged actual velocity and ν_t . It can cause divergence when the trained NN is used to predict ν_t as in the prediction loop, as the initial velocity field is different from the converged velocity field. The values of ν_t obtained using the velocity field of the initial iteration as an input to NN can steer the solution away from the convergence. Hence, the potential velocity field is a good alternative as it is easily computable and utilizes initial and boundary conditions that remain constant throughout the simulation. Thus, using potential velocity eliminates the need to retrain the NN every iteration (unlike the actual velocity field) in the training loop while also facilitating the convergence of the prediction loop.

Another possible alternative is to use boundary conditions solely as input to the NN since they generally remain constant throughout

the simulation. The problem with using boundary conditions is the input to NN lacks information about the internal domain. The learning algorithm can use this information about the interior to identify any underlying relationships, facilitating its training. Therefore, using only boundary conditions as input can make training difficult because the NN has to learn to associate the boundary values with the intricate flow and geometric features that affect ν_t . On the contrary, the potential velocity field, which represents approximate flow features across the domain, also encapsulates the geometric information about the domain.

A. Backward-facing step

For testing algorithm 2, an NN is trained to predict ν_t for flow over a backward-facing step. The details of implementation for the backward-facing step are derived from the NASA turbulence database.⁵⁶ Figure 7 shows a schematic of the domain; $H = 0.0127$ m is the height of the step. The total height of the domain is set as $9H$. The boundary conditions for velocity are no-slip at walls and zero gradients at the outlet. For pressure, a zero gradient is used at the inlet and walls, with a fixed value of zero at the outlet. The ν_t is constrained using the Spalding wall function at the walls. The viscous terms use a central difference scheme for discretization, and the advective terms use the linear-upwind stabilized transport discretization scheme. The loss function provides the freedom to choose any discretization scheme from multiple possibilities available in the OpenFOAM suite.

For validating the NN solver, the results ($Re = 36\,000$) from NN solver are compared with the corresponding benchmark CFD solver and NASA database (CFL3D-SA).⁵⁶ Figure 8 compares the coefficient of pressure (C_p), skin friction coefficient (C_f), and velocity (u_x/U_∞). Here, u_x is the x-component of the velocity, and U_∞ is the inlet velocity. C_p and C_f are plotted along the bottom wall starting downstream of the step ($x = 0$ in Fig. 7), and u_x/U_∞ is plotted at the probe location (P_n in Fig. 7) for $n = 1, 4$, and 10 . From the C_f plot in Fig. 8, the reattachment length (R_t) is $6.1H$ for NN and CFD solver and NASA database (CFL3D-SA).⁵⁶ Additionally, the results for both NN and CFD solver agree well with the CFL3D-SA for each of C_p , C_f , and u_x/U_∞ . However, the current CFD and NN solvers have a slight difference in C_f (downstream the reattachment point) compared with CFL3D results. This difference is due to the use of a wall function in the present study as opposed to CFL3D.⁵⁷ The resulting accuracy from current mesh and solver settings is acceptable as both NN and CFD solvers produce similar trends to the benchmark CFL3D case. To reiterate, upto this point, the NN solver is used to solve flow for single Re ; therefore, the NN does not have any prediction capability for arbitrary Re . The NN is trained for prediction at unseen Re for the backward-facing step in Sec. V A 1.

The NN network is a five-layered, fully connected network. Each of the three hidden layers has ten neurons with \tanh as an activation function. The input and output layer scale with the mesh and have 3600 neurons each. The mesh size was selected by performing grid independence studies up to a size of 25 000 cells. The grid is designed for $y^+ \sim 30$ because of the wall function, significantly reducing the required grid size. The total number of trainable parameters in the NN is 75 830. As explained in the Sec. II A, the focus of the present study is on the loss functions. Therefore, no NN optimization study was performed.

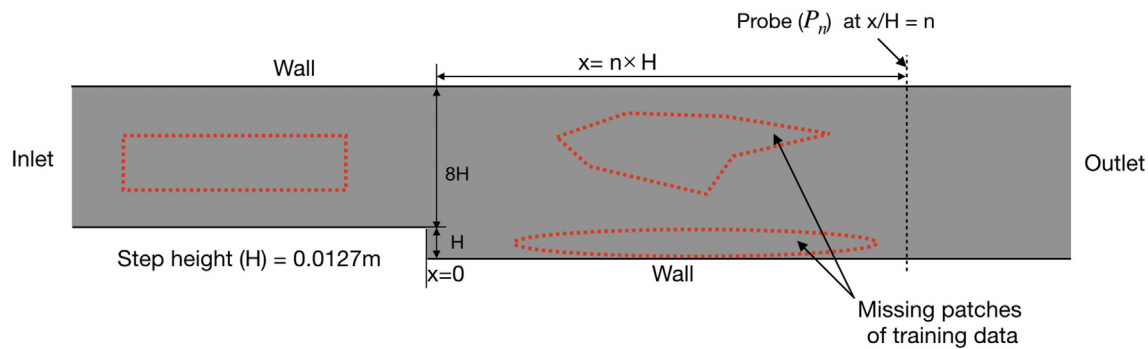


FIG. 7. Schematic of the backward-facing step. Here, H is 0.0127 m, and dotted line representing given by P_n is used as probe locations for plotting velocity and ν_t data at location $x/H = n$. The dotted red patches show the missing data at each data point of the training dataset for validating hybrid training.

The number of epochs reduced linearly with the number of SIMPLE iterations; that is, at the start of the training, 300 epochs were used, which were reduced by five at every tenth SIMPLE iteration before settling to a minimum number of 20. The initial learning rate for n th SIMPLE iteration is given by $(1 \times 10^{-3}) / (1.2 \times (n - 1))$; that is, at the start of the first SIMPLE loop, the learning rate was 1×10^{-3} , which reduced to 8.3×10^{-4} at the start of the second SIMPLE iteration. In addition to reducing the learning rate globally at the start of each SIMPLE loop, the learning rate was reduced locally as well with the passing of epochs. Here, the local learning rate was reduced by a factor of four after every 30% of the corresponding number of epochs passed.

1. Training NNs for prediction of ν_t

The NN is trained using both hybrid [Eq. (7)] and fully unsupervised [Eq. (5)] loss functions. In hybrid learning, each instance of the training dataset has some random patch missing from the ν_t field, as shown in Fig. 7 (dotted red patches). Four different training data sets,

each missing a varying percentage of training data (ν_t) across the solution domain, were used to test the hybrid learning methodology. The missing data comprised 20%, 40%, 50%, and 70% of the solution domain for every ν_t field in training data.

The algorithm 2 is run in a batch-based mode (batch size 5) to training the NN. Motivated from the work of Maulik *et al.*,⁵⁵ the training is done at five Re s equally spaced between Re 30 000 and 36 000. Figure 9 shows the comparison of maximum loss value vs the number of training iterations (SIMPLE iterations) for hybrid and fully unsupervised learning. In addition, Table I outlines the training speedups corresponding to each training dataset. For instance, in the case of a dataset with 20% of data missing, the hybrid loss function reaches the loss threshold of 4.47×10^{-7} in 374 training iterations, approximately five times faster than the 1886 iterations for fully unsupervised learning. The speedup is due to the ability of hybrid loss functions to extract information from partially available data and patch it with physics-based training, which is targeted only at the locations that are missing the data. Intuitively, the speedups diminish as the percentage of missing data is increased. However, even in the case with 70% of the

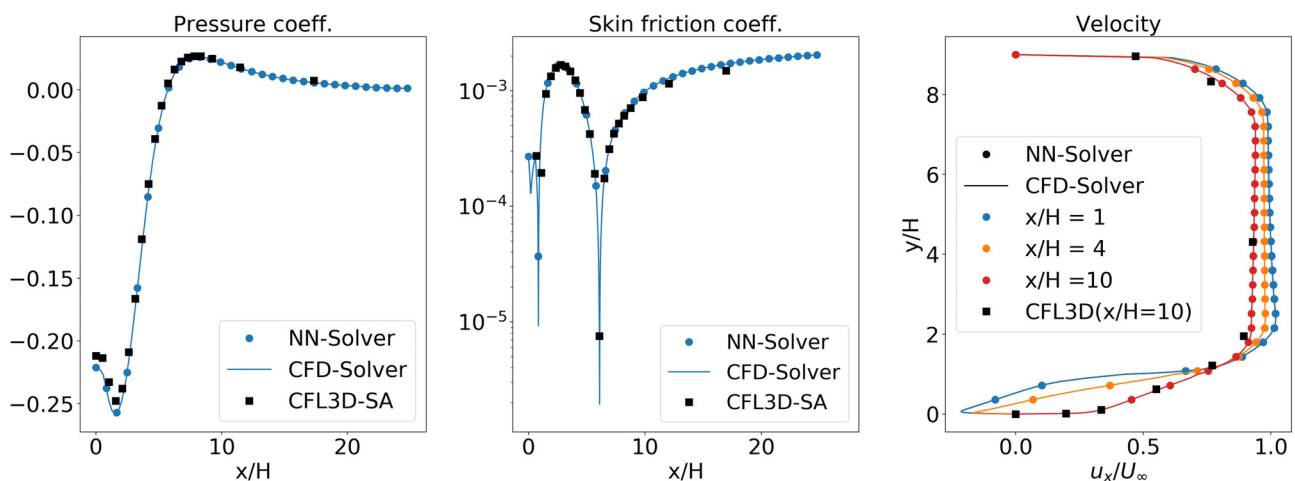


FIG. 8. Comparison of results from NN solver with CFD solver for C_p , C_f and u/U_∞ at $Re = 36\,000$. For C_f and C_p , the values are plotted along the bottom wall downstream of the step, while u/U_∞ is plotted at probe P_n (Fig. 7) at $x = n \times H$, $n \in \{1, 4, 6\}$. The benchmark solution for the case can be also found in NASA turbulence repository for Spalart–Allmaras model (CFL3D-SA).⁵⁶

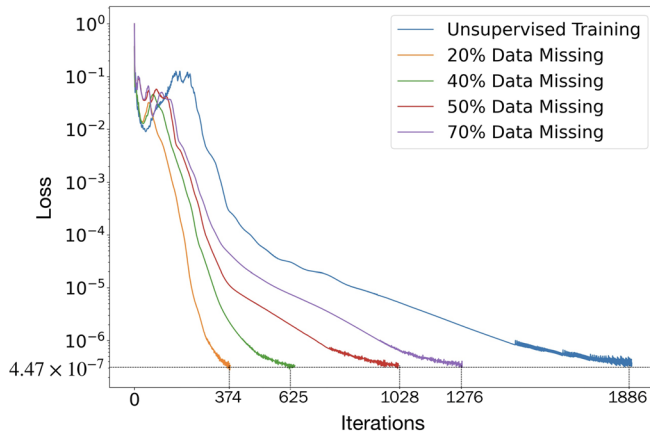


FIG. 9. Comparison of hybrid and fully unsupervised/data-free training in terms of mean absolute loss vs training iterations (SIMPLE). For all the cases, the hybrid approach successfully built upon the partially available data and trained every NN to an error threshold of 4.47×10^{-7} in lesser iterations compared with fully unsupervised learning (1886 iterations). However, speedups diminish as the percentage of missing data increases. Supervised learning cannot be applied here due to missing data.

domain having information missing, the hybrid loss function extracted marginally available data and leveraged it to provide a speedup of 1.5 times compared with unsupervised learning.

Nevertheless, hybrid learning should be viewed only as a particular case, not as an alternative to unsupervised learning. Data-free learning has its advantages, a major one being complete independence from the training data.

2. Validation of ν_t prediction accuracy of trained NN

In this section, the prediction accuracy of the trained NN is validated against the CFD solver. It was observed that the predicted ν_t from NNs trained from data-free and hybrid approaches were mutually in close agreement due to being trained up to similar loss thresholds. Hence for brevity, only the prediction from the NN trained via a data-free method is compared with the benchmark CFD solutions. Figure 10 compares the ν_t profiles for NN prediction with those calculated using the PBiCG solver at Re 33 900. For further analysis, Figs. 11 and 12 compare NN prediction with CFD solver for ν_t and u/U_∞ ,

TABLE I. Training speed for hybrid learning vs fully unsupervised learning to reach training error (mean absolute error) of 4.47×10^{-7} .

Percentage of data missing at each data point over entire dataset	Training iterations to reach MAE of 4.47×10^{-7}	Training speedup vs unsupervised learning
20%	374	5.04
40%	625	3.01
50%	1028	1.83
70%	1276	1.47
Fully unsupervised	1886	NA

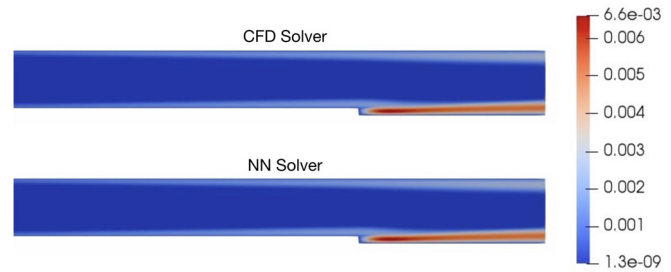


FIG. 10. Comparison of ν_t contours at Re 33 900 for flow over a backward-facing step solved by CFD solver (PBiCG) and predicted by NN solver, respectively.

respectively, along the probe P_n located at $n = 1$ and six downstream of the step. The maximum deviation between the calculated and predicted values is under 1%. The L-2 error norm between predicted and calculated values of ν_t at each Re is less than 3.52×10^{-8} . The same at the boundaries is less than 3.79×10^{-8} .

It should be noted that NN inputs and outputs used in this study are simplistic and suitable for validating physics-based learning in the present context due to the high accuracy obtained in current applications. However, large gradients can render NN training difficult and inaccurate for high Reynolds number flows.^{10,58,59} Therefore, for highly specialized applications (high Re), the current approach must be integrated with specialized techniques such as reduced order modeling,⁸ zone-based,⁵⁸ and feature-based^{10,59} learning to facilitate and accelerate the NN training. Here, an additional analysis of application-specific feature selection needs to be performed and coupled with the presented learning methodology. In this case, the feature-based training can help improve the accuracy of the NN with a reduction in the number of input features⁵⁹ and improving generalization.¹⁰

3. Solution speed for the NN approach

The proposed approach provides significant speedups (up to 6.8 times) in the solution times as compared to the conventional method using the PBiCG solver. The overall speedups for reaching the solution are shown in Table II. It should be noted that for determining solution times, the time taken to obtain the potential flow solution has also been taken into account.

The observed speedups have been previously attributed to the ability to use larger relaxation factors for pressure (α_p) and velocity (α_u) in the case with predicted ν_t as compared to the ones used in the case where turbulence closure PDE needs to be solved.⁵⁵ An analysis independent of relaxation factors was performed to gain further insight into observed speedups.

Both solution methodologies (CFD and NN solver) were employed with identical relaxation factors, namely, 0.7 for pressure and 0.9 for momentum equation. The relaxation factor (α_{ν_t}) used for solving turbulence closure (SA model) was selected as 0.35. Figure 13 shows the residuals of the velocity and pressure at Re 33 900 for given values of α_{ν_t} , α_p , and α_u . It can be seen that using the same relaxation factors, the residuals for the NN solver are decreasing at a higher rate, that is, converging in fewer iterations. Thus, it can be concluded that other effects help accelerate the solution, in addition to (larger) relaxation factors and obviating the computational effort associated with the

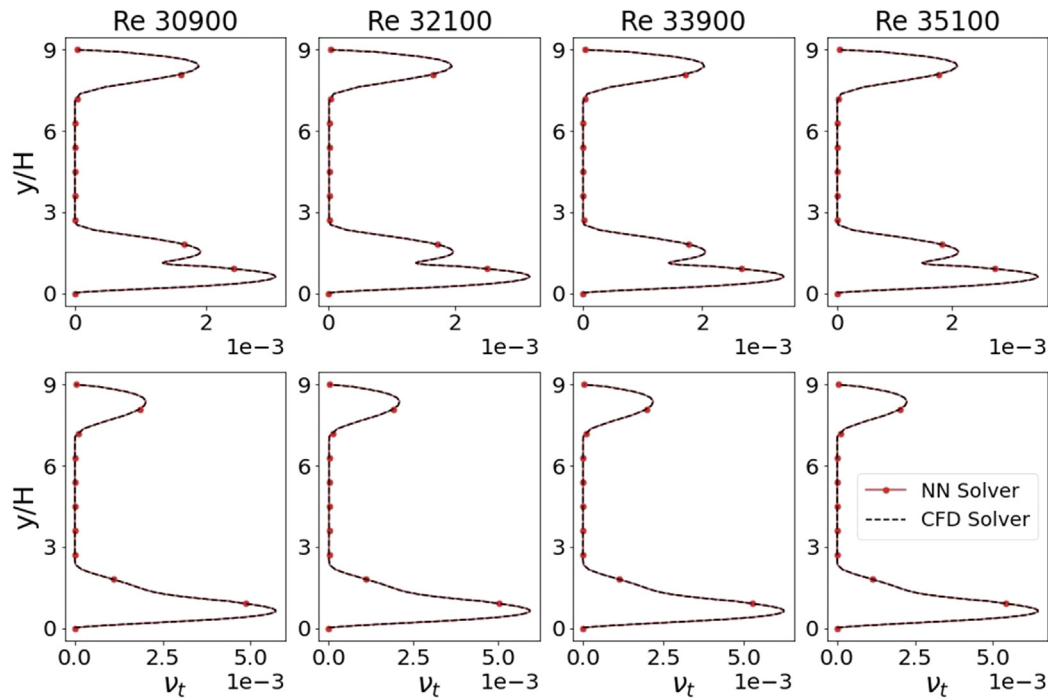


FIG. 11. Line plots comparing the values of ν_t at Re 30 900, 32 100, 33 900, and 35 100 for NN solver (red) vs CFD solver (black, PBiCG). The first row shows the ν_t at probe P_1 (Fig. 7), that is, $x = H$; second row plots the values at P_6 (Fig. 7) corresponding to $x = 6H$ downstream of the step.

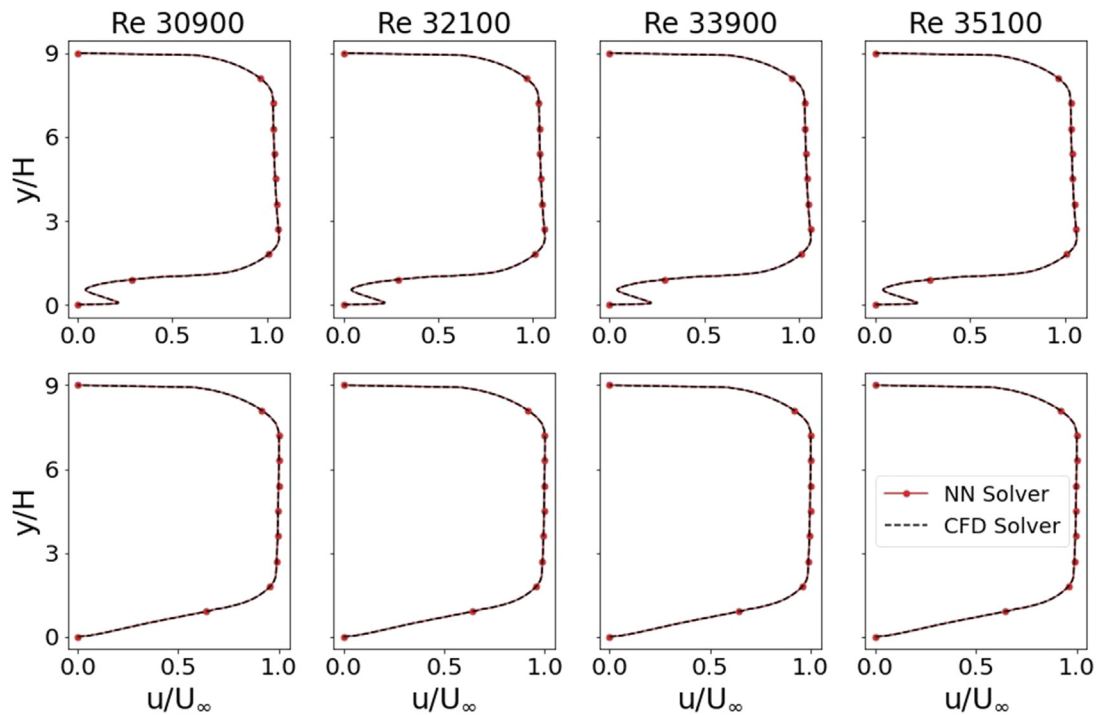


FIG. 12. Line plots comparing the values of u/U_∞ at Re 30 900, 32 100, 33 900, and 35 100 for NN solver (red) vs CFD solver (black, PBiCG). The first row shows the u/U_∞ at probe P_1 ; second row plots the values at P_6 (Fig. 7).

TABLE II. Comparison of overall solution times for neural network-based solver with a PbiCG solver for turbulent eddy viscosity (ν_t) in a SIMPLE loop. All calculations are done with serial execution on an Intel Xeon(R) E5520 with a clock speed of 2.26 GHz.

Re	Solution time (s)		Speedup ratio
	Neural network	PbiCG	
30 900	2.77	18.68	6.74
32 100	2.78	18.85	6.78
33 900	2.76	19.02	6.89
35 100	2.82	18.72	6.63

solution of the SA model. However, a comprehensive analysis, considered beyond the scope of this study, is needed to fully understand the complex interaction between the predicted ν_t and pressure and velocity solvers. Also, a scaling analysis of problem size vs speedup would provide further insight into the convergence mechanics of NN solver.

It should be noted that other methods can also speed up the solution, such as the use of multi-grid solvers or preconditioners. However, the NN approach here is not presented as a substitute but rather as an enhancement to those methods. The current approach can be used to further augment the speedups for any of the methods above. Additionally, for high Re flows, the current input and output of the NNs can affect the speedups. Therefore, specialized techniques such as reduced order modeling,⁸ zone, and feature-based NNs^{10,58,59} can be used with the current methodology to take into account the effect of large gradients and optimally scale the speedups for high Re flows.

VI. CONVECTIVE HEAT TRANSFER

This section uses an NN to predict the temperature for a steady-state convective heat transfer problem. The loss function, in this case, is the constant-property energy equation given as

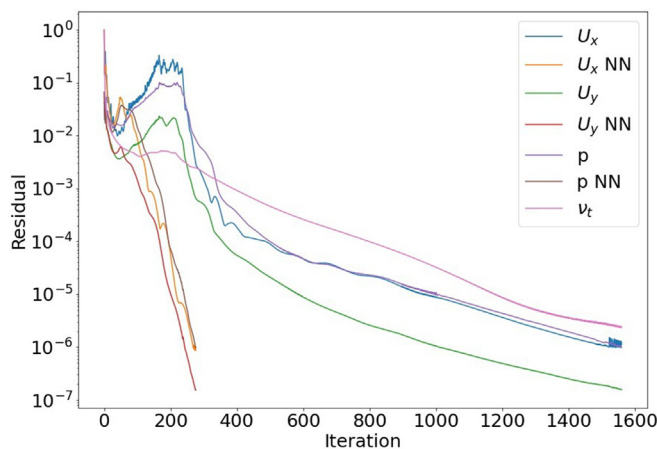


FIG. 13. Residual plots for NN-based vs conventional solver for comparing convergence speed at Re 33 900. Residuals for the NN-based solver decay at a faster rate (6.89 times) than conventional solvers using the same relaxation factors.

ALGORITHM 3: Modified SIMPLE algorithm to train an NN for a convective heat transfer problem.

- 1: Calculate the potential velocity field (\mathbf{u}^p) for the given boundary conditions.
- 2: Solve for \mathbf{u} and \mathbf{p} using SIMPLE algorithm.
- 3: **if** Training NN **then**
- 4: Discretize the given heat transfer model to be posed as physics-based loss function. For this study it is Eq. (10).
- 5: Predict the temperature from NN, with potential velocity (\mathbf{u}^p) as an input and substitute it to loss function to obtain the deviation at each grid point.
- 6: Feedback the NN with the loss values for training.
- 7: **if** Prediction from trained NN **then**
- 8: Predict the temperature from NN, with potential velocity as an input.
- 9: **if** Additional effects of temperature are present **then**
- 10: Before next iteration, solve for effect of temperature on flow parameters such as viscosity and pressure.
- 11: Go to step 2 until convergence.

$$\nabla \cdot (\mathbf{u}T) = \alpha \nabla^2 T, \quad (10)$$

where T is the temperature field, and α is thermal diffusivity. In Eq. (10), pressure work, Dufour effects, and viscous dissipation are all neglected. As shown in algorithm 3, in the training loop, the NN is used to predict the temperature field, which is subsequently fed to the loss formulation to provide the residual information for training the NN. For the prediction loop, a trained NN is used to predict the temperature field, T for the given flow conditions by using the potential velocity field as an input to NN.

A. Flow across a heated cylinder

The NN-based solver is used to compute the temperature distribution around a heated square cylinder placed in a fluid flow (Fig. 14). The flow is steady and confined, with a blockage ratio of seven. The domain measures $20H$ and $7H$ in stream-wise x and cross-wise y directions. The center of the cylinder is located at a distance of $5.5H$

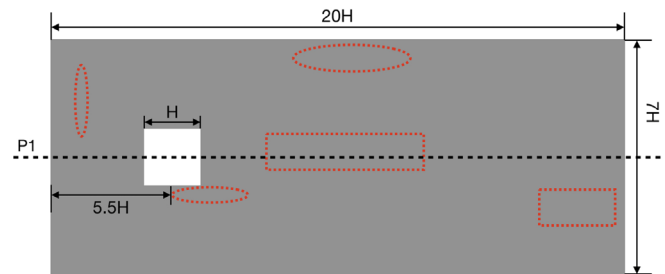


FIG. 14. Schematic of the domain for fluid flow and heat transfer around a heated square cylinder. Here, $H = 0.05$ m. The dotted red patches showing the missing data patches for supervised training at each data point of the training dataset. The centerline, P1, is used as a location for plotting temperature data.

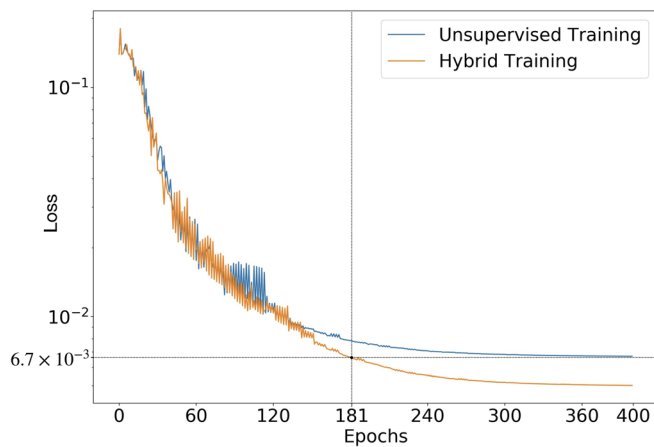


FIG. 15. Comparison between hybrid and fully unsupervised training in terms of mean absolute loss/error value as a function of training epochs. The hybrid loss functions reach the loss value of 6.7×10^{-3} in 181 epochs which is 2.2 times faster than corresponding 400 epochs of fully unsupervised learning.

from the inlet. Here, $H = 0.05$ m is the length of one side of the square.

Similar to Sec. V A, the proposed framework is validated for training the NN using both hybrid and fully unsupervised loss functions. As shown in Fig. 14 (dotted red patches), here also, each data point of the training dataset has some random patch of temperature field missing. The velocity boundary conditions are no-slip along with the cylinder and outer walls. The cylinder surface temperature is 350 K, while the walls and inlet fluid are at 300 K. The Prandtl number, in this case, is 1.1. As described earlier, the input to the NN is a potential velocity field.

1. Training NNs for temperature prediction

Figure 15 shows a plot comparing the normalized maximum loss value (across the domain) for hybrid and fully unsupervised learning methodologies. The missing instances in each training data point comprised 30% of the solution domain. Algorithm 3 is run for a batch of five Re for training, namely, 10, 20, 30, 40, and 50. The NN, in this case, has five layers with three hidden layers of ten neurons. The activation function used is \tanh . The input and output layers have a size of 3770, each making total trainable parameters 79 400. The initial

learning rate is 1×10^{-3} and decreases by a factor of 1.2 after every ten epochs. As mentioned above, NNs in this study are not subjected to any optimization as the focus here was on loss functions. Both fully physics-based and hybrid losses train NNs for 400 epochs using similar hyper-parameters. The hybrid strategy trains NNs faster (i.e., lower loss value) for the given number of iterations. As shown in Fig. 15, the accuracy reached by fully supervised learning in 400 iterations is achieved by hybrid learning in 181 epochs; that is, a training speedup factor of 2.2 is achieved by hybrid learning.

2. Predicting temperature using trained NN

The solutions are predicted for steady-state flow at Re of 35 and 45. Similar to Sec. V A, the predictions of NNs trained using hybrid and fully unsupervised loss functions were in close agreement. The following results are only for the NN that was trained by using the hybrid loss function. Figure 16 shows the comparison of non-dimensional temperature (T^*) contours from the NN and CFD solver at Re of 35. Here, T^* is given as

$$T^* = \frac{T - T_{min}}{T_{max} - T_{min}}, \quad (11)$$

where T is predicted temperature, $T_{max} = 350$ K (cylinder surface), and $T_{min} = 300$ K (inlet fluid). Figure 17 provides further insight into prediction accuracy by plotting comparisons of the predicted and calculated (PBiCG solver) temperature along the centerline of the domain (P1 in Fig. 14) at Re of 35 and 45. The NN predicted results match well with those of the calculated solution. The deviation observed between the predicted and calculated results is less than 1.82%. The L2-norm error for the entire domain averaged for both predicted Re is 0.006 31.

It should be noted that the flow model considered in this case assumes that there is no impact of the temperature field on the variables such as viscosity. However, suppose such an effect was present, additional equations outlining said effects must be solved to resolve the flow field accurately. The modularity resulting from integrating NNs and CFD algorithms allows for additional equations to be solved simultaneously, with current equations as shown in Sec. VII.

VII. SIMULTANEOUS SOLUTION OF POISSON (PRESSURE) AND ENERGY EQUATION

As mentioned in Sec. VI, the modular nature of the present approach can be used to solve multiple equations simultaneously. In this section, multiple steps of the SIMPLE algorithm are replaced with

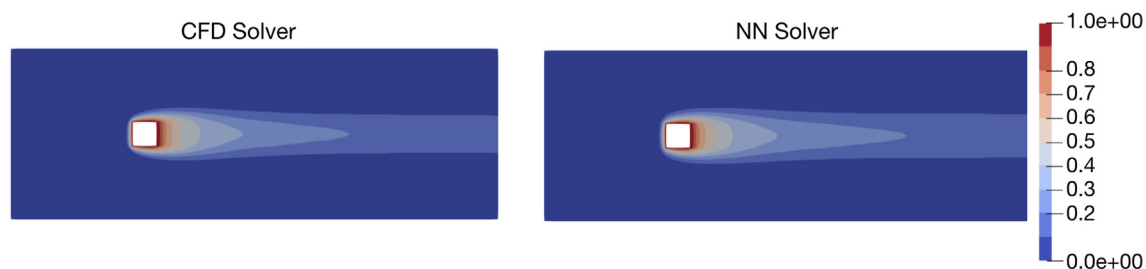


FIG. 16. Comparison of temperature [T^* , Eq. (11)] contours for the flow around a heated cylinder at Re 35. Left: Calculated using CFD solver (PBiCG), Right: Predicted using NN.

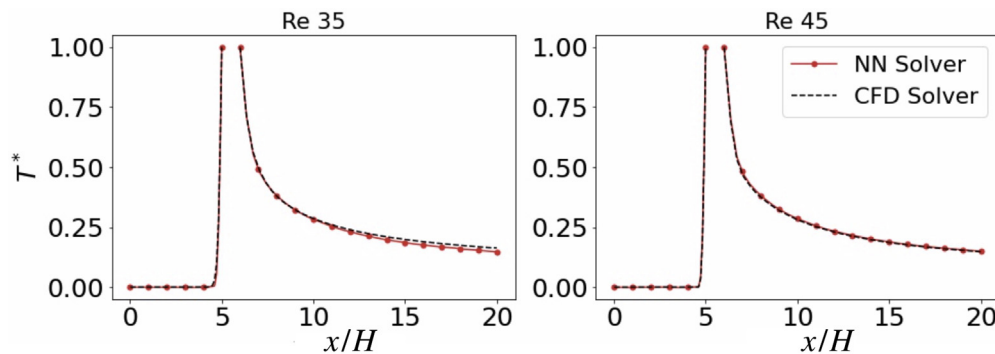


FIG. 17. Centerline plots for temperature (T^*) predicted by NN solver (red) and that of CFD solver (black, PBiCG) along the line P1 in Fig. 14. Here, T^* is obtained using Eq. (11).

NNs. Hence, all the corresponding governing equations must be simultaneously posed as loss functions. Here, the NNs replace the pressure and temperature calculation steps; that is, Poisson and energy equations are posed as loss functions. Similar to the lid-driven cavity case, the NNs here are only used as solvers to quantify the accuracy of the NN-based solution for pressure and temperature.

The case considered is steady laminar flow over a 3D backward-facing step. The computational domain has length, breadth, and height of $12H$, $2H$, and $2H$, respectively. The entrance region has a length of $2H$, where $H = 0.1$ m is the step height. The lower surface of the entrance region is at 350 K, with the entering fluid and remaining surfaces at 300 K. The Prandtl number, in this case, is 1.1. The NNs for pressure and temperature solution have architecture and hyperparameters similar to the one used in Sec. V A. The total trainable parameters for NNs, in this case, are 98 930, with 4700 nodes in input and output layers. Solutions are obtained at Re of 40, 50, and 70 using the potential velocity field input to the NN. Figure 18 shows the mean absolute loss vs iterations for the NN solver.

Figure 19 shows a comparison of the velocity (u/U_∞), pressure field ($P/\rho U_\infty^2$), and temperature [T^* , Eq. (11), $T_{max} = 350$ K and $T_{min} = 300$ K] profiles obtained from CFD and NN-based solvers at $Re = 70$. Here, u is the velocity magnitude, P corresponds to pressure,

and U_∞ is inlet velocity. From the streamline plots in Figs. 19(g) and 19(h), the NN solver can resolve the primary re-circulation zone correctly. For further evaluation, temperature, pressure, and velocity results for NN solver vs CFD solver are compared in Figs. 20 and 21 along the centerlines shown as P1 and P2 in Figs. 19(a) and 19(c). Also, the reattachment lengths of NN-based solutions are within 0.3% of the calculated results. The averaged L-2 error norm for all Re is 4.8×10^{-5} for pressure and 0.007 91 for temperature.

VIII. DISCUSSION AND KEY CONTRIBUTIONS

A. Flow-specific discretization

The discretized loss functions presented in this study allow for flow-specific discretization, which is similar to conventional CFD applications. The specificity of loss functions is realized by using the different discretization schemes for the presented flow problems. The lid-driven cavity used a central difference scheme, while the backward-facing flow used linear upwind stabilized transport to discretize advective terms. Hence, the framework provides the freedom to perform any flow-specific modification to loss functions based on CFD know-how. However, discretization schemes applied in this study are not claimed to be ideal for the corresponding flows. The focus here was to demonstrate the flexibility of using any discretization scheme from multiple options available in the OpenFOAM library.

B. Hybrid loss functions

Hybrid loss functions presented in this study combine the elements of supervised and unsupervised learning for training the NNs in situations where the data presented are partially available. As shown in the lid-driven cavity flow, the hybrid loss functions can extract the available information such as primary vortices from the partial observations and patch it with a physics-based loss function to generate secondary vortices (missing in the data provided). In addition, the training of NN from partial observations is demonstrated for backward-facing step and flow around a heated cylinder. The hybrid loss function proved advantageous by expediting the training compared to data-free learning. The accelerated training is attributed to the ability of hybrid learning to build upon the previous effort to generate data. These data are discarded in conventional supervised learning, wasting the resources invested to generate it.

Though the hybrid loss functions are proposed for both experimental and CFD training data, only the cases in the context of CFD

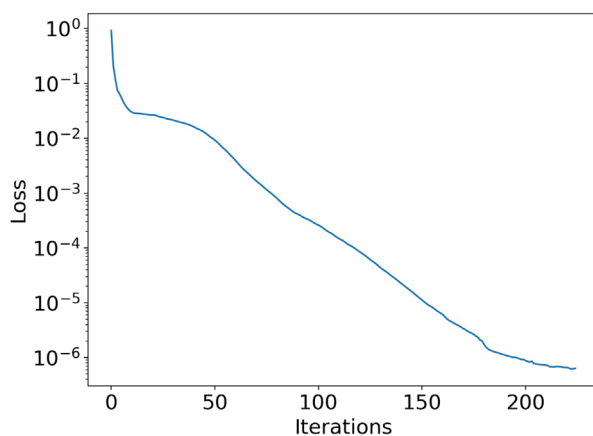


FIG. 18. Mean absolute loss vs iterations for NN solver for convective heat transfer in a backward-facing step.

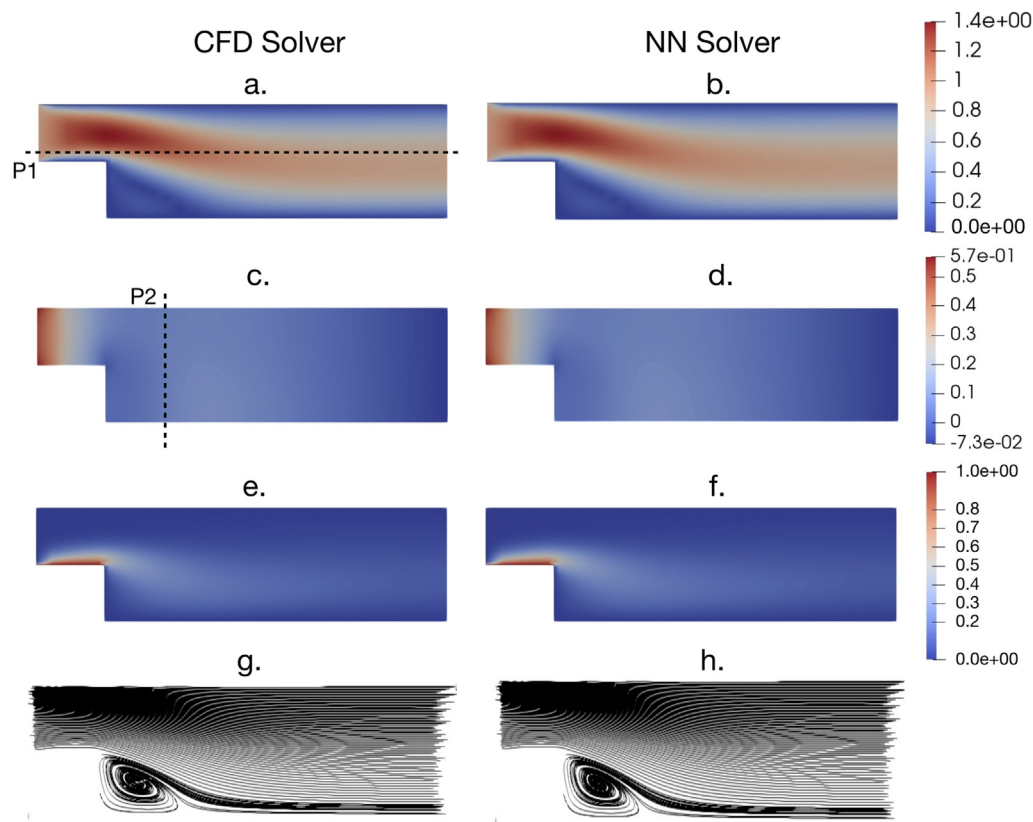


FIG. 19. Comparison between [(a) and (b)]. Velocity (u/U_∞^2) [(c) and (d)]. Pressure ($P/\rho U_\infty^2$) [(e) and (f)]. Temperature (T^*) [(g) and (h)]. Streamlines for the CFD and NN solvers along the center plane of the domain at Re 70. Here, u is the velocity magnitude, P corresponds to pressure, U_∞ is inlet velocity, and T^* is obtained using Eq. (11) with T_{min} 300 (fluid) and T_{max} 350 (bottom wall of inlet zone). N.B.: The dotted lines P1 and P2 in a and c represent the locations used for line plots in Figs. 20 and 21.

data are validated. However, the same methodology can be used to train an NN from experimental data by adjusting the location of grid points of the data-driven part of the hybrid loss function to align with the probe locations for which the data are available across the domain. By contrast, the rest of the grid points can utilize the physics-based part to train the NN where the experimental data are unavailable.

C. Generalizability

The discretized nature of loss functions aids in easy integration into CFD packages. The versatility offered by the resulting CFD-ML platform is validated in this study by solving a variety of problems involving laminar flow (lid-driven cavity), RANS turbulent closure model (backward-facing step), and heat transfer (flow around heated cylinder and backward-facing step). The only modification for each flow was posing a different governing equation as a loss function. An alternative to these CFD-ML integrated platforms would be a stand-alone ML framework such as TensorFlow, where one has to code the flow problems and any corresponding flow changes from scratch. Instead, in the present TensorFlow-OpenFOAM framework, OpenFOAM libraries handled changes in geometries, boundary conditions, and discretization schemes circumventing programing bottleneck(s).

The focus of this study was to propose and integrate physics-based deep learning into CFD. Therefore, for the simplicity of NN architecture, the present research only used steady-state flow problems to validate the proposed methodology. Unsteady flow problems require special NN architectures such as LSTMs (long short-term memory neural networks) to resolve flow variables. The future work will focus on unsteady flow problems aligning with the work of Pant *et al.*⁷ and use LSTMs for implementing physics-based learning into unsteady flows.

IX. CONCLUSIONS

The present study formulates generalized unsupervised learning as an alternative or an augmentation to supervised learning. As an alternative, unsupervised learning is solely realized through physics-informed loss functions, which require no data to train an NN. As an augmentation, the physics-informed loss functions are hybridized with data-driven loss functions to train an NN from partial or sparse observations that cannot be used with purely data-driven training. In addition, the SIMPLE algorithm is modified to accommodate the physics-based NN training by applying feedback loops at various solution steps.

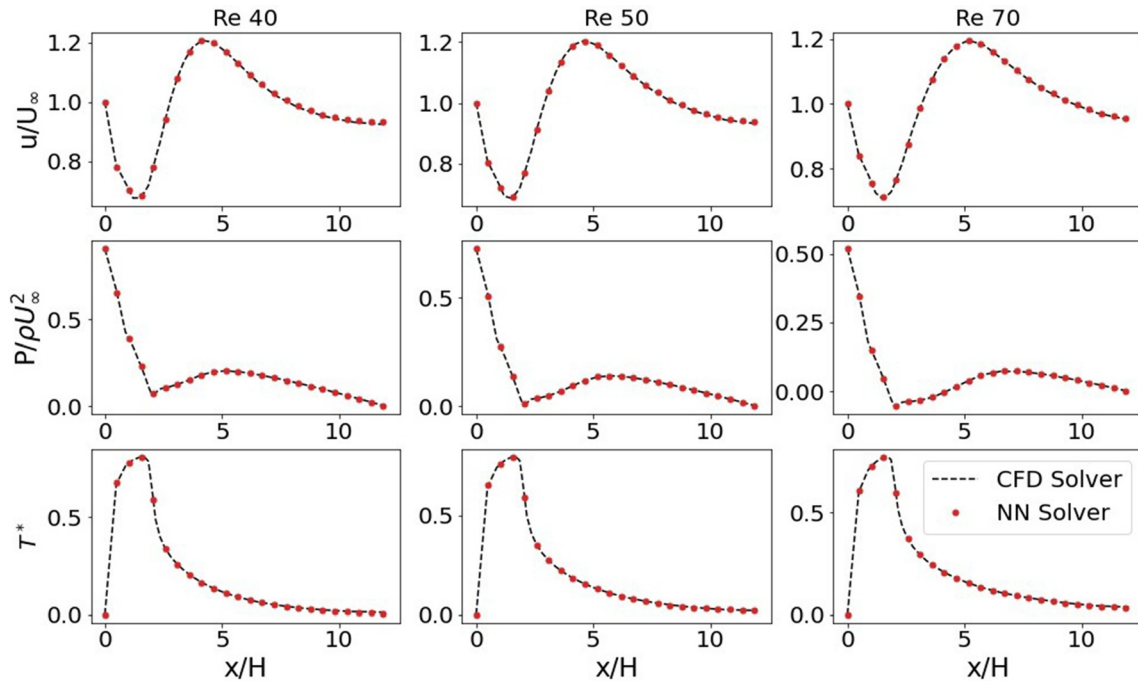


FIG. 20. Comparison between CFD and NN-based solution of velocity (u/U_∞), pressure ($P/\rho U_\infty^2$), and temperature (T^*) from Fig. 19(a) located along P1, the centerline of the domain parallel to x axis and $0.1H$ above the base of the entrance region.

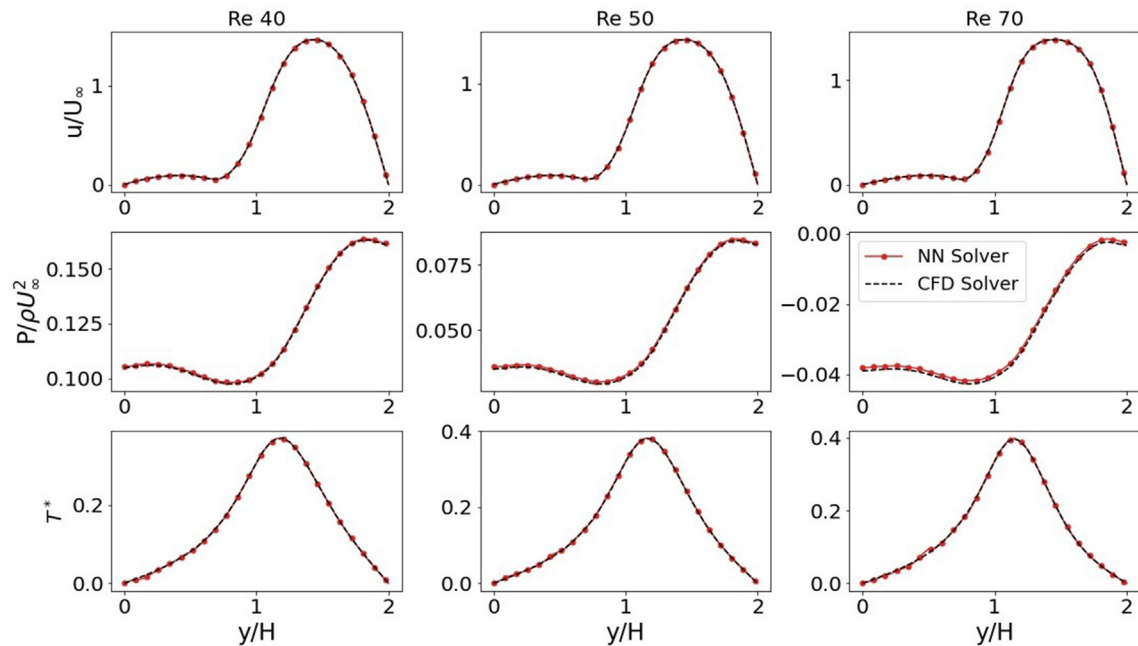


FIG. 21. Comparison between CFD and NN-based solution of velocity (u/U_∞), pressure ($P/\rho U_\infty^2$), and temperature (T^*) along probe location P2 in Fig. 19(c) located at the centerline of the domain parallel to y axis and at a distance H downstream of step.

- (i) The discretized nature allows the presented loss functions to benefit from well-developed numerical schemes of CFD to resolve different flow phenomena. Similar to conventional CFD solvers, these loss functions have access to all the numerical schemes of CFD (OpenFOAM). This access enables training NNs for any flow, which can be solved by using numerical schemes available in OpenFOAM. Though for demonstration, OpenFOAM is used here, and the presented loss functions can be used with any finite-volume-based CFD solver.
- (ii) The results predicted by NNs trained in this study agreed well with those of numerical solvers. Also, the NN solver provided significant speedups of approximately 6.7 times for turbulence modeling. These speedups are encouraging. However, a detailed analysis of the complex interactions between the governing equations and solution variables, together with a scalability analysis for high Re flows, needs to be performed in future work to get more insight into utilizing the speedups optimally. Though for simplicity, this study trains NNs on Spalart–Allmaras model, and the speedups obtained stimulate further exploration of models such as $k - \varepsilon$ and $k - \omega$ SST.
- (iii) One significant benefit of discretized functions is compatibility with the CFD algorithms and solvers. The present work uses this compatibility to create a general-purpose unsupervised learning CFD-ML platform where different flow problems can be studied by machine learning. The unsupervised learning in TensorFlow is integrated with OpenFOAM to form a CFD-ML test-bed. This test-bed allows the freedom to switch between different flows and geometries with relative ease circumventing the programming bottleneck(s) of coding every detail on a stand-alone TensorFlow application. NNs trained in this study validate this freedom on a comprehensive range of flow problems involving Poisson-like/pressure equation, SA turbulence closure model for RANS, and energy equation.
- (iv) Another contribution of this study is the hybrid learning approach, which formulates loss functions by combining supervised learning with the *a priori* in the form of governing equation. This combination was validated for training an NN from data missing multiple instances. These data cannot be used with standard supervised learning algorithms. The hybrid learning strategy recovered complex flow features such as secondary vortices from locations of missing data for the lid-driven cavity flow case. In addition, the hybrid loss function extracted the information about temperature and ν_t from the partially available data for flow around a heated square and a backward-facing step, respectively. The information extracted from the partially available data helped train the NN five times faster than purely unsupervised training.

The validation served as proof of concept and encouraged future work with the experimental data, where hybrid learning can be more apt due to the loss of data due to faulty probes. However, the governing equations used to recover the missing data in experiments can present limitations on the recovered flow phenomena. For these limitations, future work will utilize the concepts such as transfer

learning so that the hybrid learning recovering the data and training the NN can also benefit from previous training runs on datasets of the flows that resemble the flow under consideration.

The present methodology has been validated in the context of the popular SIMPLE algorithm only. However, the presented training methodologies can be integrated into other pressure-linked or numerical algorithms. In addition, the NN solvers presented in this study can be used as surrogates to numerical solvers for any other partial differential equations. However, the inputs and outputs of the NNs have to be adjusted based on the application. Since this study aims to validate the unsupervised training and its accuracy, only steady-state flows were used as test cases for simplicity. In future work, the present methodology will be used with long short-term memory networks as surrogates for predicting the solutions to unsteady flow problems.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Deepinder Jot Singh Aulakh: Conceptualization (equal); Data curation (equal); Formal analysis (equal); Investigation (equal); Methodology (equal); Software (equal); Validation (equal); Visualization (equal); Writing—original draft (equal); Writing—review and editing (equal). **Steven B. Beale:** Funding acquisition (equal); Investigation (equal); Methodology (equal); Project administration (equal); Resources (equal); Supervision (equal); Writing—original draft (equal); Writing—review and editing (equal). **Jon G. Pharoah:** Conceptualization (equal); Data curation (equal); Funding acquisition (equal); Investigation (equal); Methodology (equal); Project administration (equal); Resources (equal); Software (equal); Supervision (equal); Validation (equal); Visualization (equal).

DATA AVAILABILITY

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

APPENDIX A: SIMPLE ALGORITHM

The semi-implicit method for pressure-linked equations (SIMPLE)³⁷ is a widely used CFD algorithm for iteratively solving Navier–Stokes equations. At the start of a SIMPLE iteration (step 1), the velocity field (\mathbf{u}) is obtained by solving the momentum equations. Here, the pressure gradient is calculated from the pressure distribution (P^{old}) from the previous iteration or an initial guess. A Poisson's pressure equation is solved for conservative pressure distribution (P^{p}) in step 2. The velocity field (\mathbf{u}) used to form the pressure equation is obtained from the first step. The pressure field (P^{p}) obtained here is under-relaxed in step 3 to obtain P^{new} used to calculate the conservative mass flux for correcting the velocity field in step 4. The iterations are repeated until convergence. However, it should be noted the SIMPLE algorithm presented here is based on OpenFOAM. The original SIMPLE algorithm by Patankar and Spalding³⁷ calculates the pressure correction (P') instead of directly obtaining the pressure (P^{p}).

ALGORITHM 4: SIMPLE algorithm³⁷ (simpleFOAM³⁸).

- 1: Obtain approximate velocity field (\mathbf{u}) by solving momentum equation. The pressure (P^{old}) used in this step is either from the previous iteration or an initial guess. The velocity here is not divergence free.
- 2: Solve Poisson equation for pressure distribution (P^{p}). This equation is formulated using the velocity field obtained in step 1.
- 3: A new set of conservative mass fluxes is calculated using pressure P^{new} which is obtained by under relaxing the pressure using $P^{\text{new}} = P^{\text{old}} + \alpha_p(P^{\text{p}} - P^{\text{old}})$.
- 4: Correct velocities using P^{new} .

For the applications presented in this study, in a SIMPLE loop, the solution of momentum and pressure equations takes 47% and 53% of the total computation time, respectively. For momentum, pressure, and energy equations in convective heat transfer, the distribution is 33%, 51%, and 16%, respectively. If turbulence is added, the distribution is 30%, 50%, and 20% for momentum, pressure, and eddy viscosity solution. However, the contribution of each equation to the total solution time should not be viewed as the sole indicator of speedup that can be achieved by coupling with NN. The number of iterations to convergence is also equally important. As seen from the backward-facing step, in addition to replacing the solution of eddy viscosity, the NN prediction also reduced the number of iterations to reach convergence.

APPENDIX B: IMPLEMENTATION OF PROPOSED LEARNING APPROACHES

1. Fully physics-based learning

Figure 22 shows the schematic implementation of fully physics-based learning onto TensorFlow and OpenFOAM integration. The schematic is divided into two parts by a dotted line. The top half of Fig. 22 corresponds to the TensorFlow implementation, and the bottom half corresponds to OpenFOAM calculations.

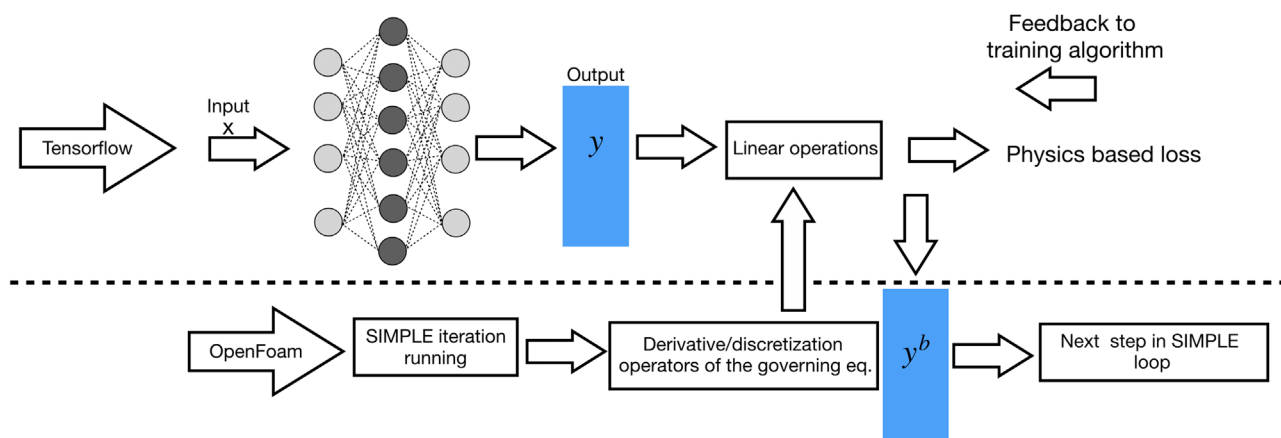


FIG. 22. Schematic implementation of fully physics-based loss onto consolidated ML-CFD platform. The dotted line separates the operations performed in TensorFlow from the ones in OpenFOAM. The solution field y is predicted and improved by using the feedback from the discretized governing equation.

The bottom half runs the SIMPLE algorithm and interacts with the TensorFlow for training a NN surrogate. The loss function for training the NN can be any governing equations present in the SIMPLE loop. These governing equations were Poisson's, Spalart–Allmaras, and energy equations for this study. First, the discretization operators for governing equation of interest are imported into TensorFlow. Here, the prediction y from an untrained NN is fed into the discretization operators to calculate the loss after linear operations such as adding/subtracting the discretized terms. The prediction y can also be referred to as the proposed solution of the governing equation. For example, y will correspond to the temperature if the energy equation is imported into TensorFlow. Then, the calculated loss is backpropagated for training the NN. Afterward, an improved prediction y^b is fed back into the OpenFOAM to be used in the subsequent calculation steps of the SIMPLE algorithm. The details about feeding y^b into SIMPLE algorithm are outlined in algorithm 1, 2, and 3, where y^b corresponds to pressure, eddy viscosity, and temperature, respectively. It should be noted that the schematic only shows the training corresponding to one predicted field for simplicity. However, a batch of fields is used to calculate the loss to realize the generalized training for the NNs.

2. Hybrid learning

As explained in Sec. II B, the hybrid loss function combines physics-based learning with the data-driven loss for training the NNs from partially available/sparse data. Figure 23(a) outlines the schematics of the hybrid loss function and its integration into CFD. Similar to fully physics-based loss, the schematic here is also divided into TensorFlow and OpenFOAM sections. The integration of TensorFlow with OpenFOAM remains identical to the fully physics-based case. However, the TensorFlow part is modified to accommodate the feedback from the available data and patch it with the physics-based loss function. The schematic of the partially available data is given by Fig. 23(b).

As shown in Fig. 23(a), the locations where data are available are trained using the data-driven loss function. On the contrary, the

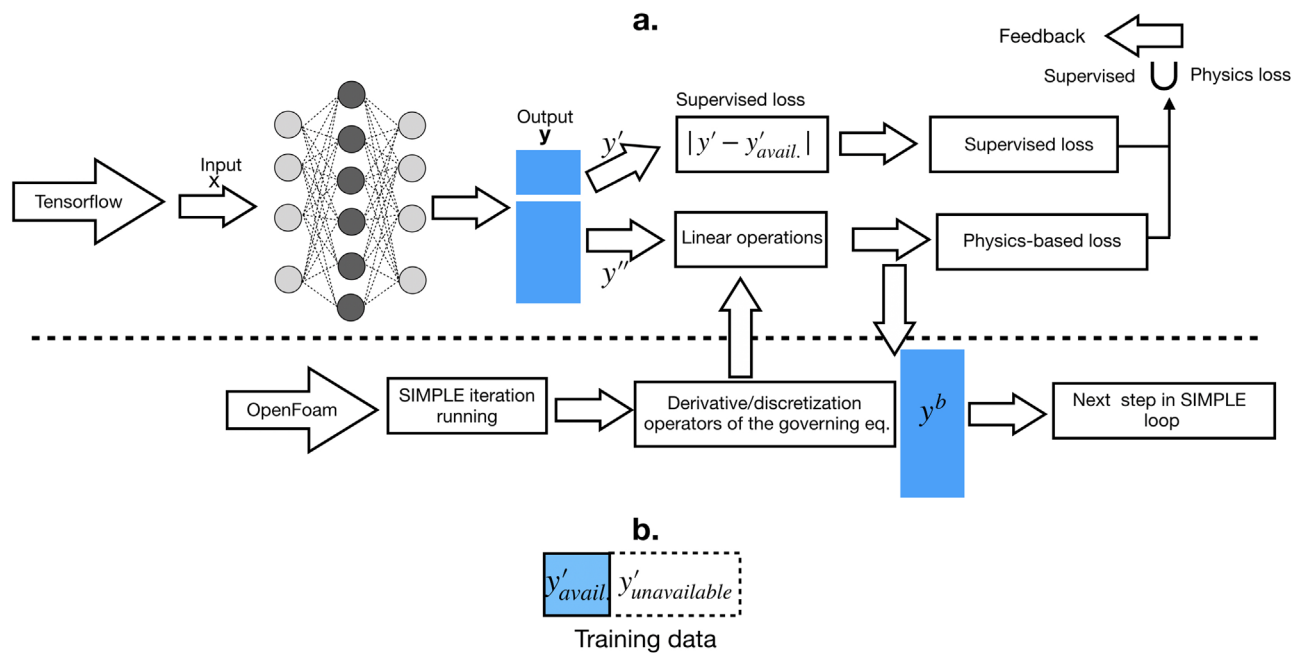


FIG. 23. Schematic implementation of hybrid learning methodology. (a) Shows the implementation details of the hybrid loss function. Here, the data-driven and physics-based losses are combined to form the feedback for the training algorithm. (b) The simplified representation of the partially available data.

parts where no labeled data are available are trained using a physics-based loss function. Before feeding into the training algorithm, both the physics and data-driven losses are combined to create feedback for the entire domain. Similar to Appendix B 1, batch-based loss is also required to infuse generalizability into the NN training.

APPENDIX C: LOSS EVALUATION FOR PRESSURE EQUATION AND COMPARISON TO PINNS

Figure 24 compares the loss evolution of fully physics-based and hybrid loss functions. The fully physics-based loss function converges in 35 training iterations compared to 133 of hybrid loss. In contrast to the turbulence closure and energy equation, the hybrid loss function converges in significantly more iterations than fully physics-based loss.

Further analysis revealed the reason behind this exception as flux imbalance and divergent non-linearity in momentum equation. The non-linearity results from the multiplication of flux with the velocity. In the case of fully physics-based loss, at every iteration of the SIMPLE algorithm, the velocity field (maybe un-converged) obtained from the momentum equation is corrected by the corresponding pressure field (un-converged). Subsequently, the said pressure and velocity field are used to calculate the flux, which conserves mass. The pressure and velocity fields mentioned above, though un-converged, are mutually compatible for getting mass conservation. This convergent mass flux is responsible for obtaining a more accurate (compared to the previous iteration) velocity field in the next iteration by solving the momentum equation. However, the partially available pressure for hybrid learning is inconsistent

with the corresponding velocity fields at each iteration. Therefore, the resulting flux is not mass conserving, which delays the convergence for the hybrid loss function. This flux imbalance also explains the wavy nature of the hybrid loss in Fig. 24. This fact is further proved by an additional analysis in which the converged flux is used along with partially available pressure in a SIMPLE loop. The hybrid loss function, in this case, converged in 25 iterations, as shown in Fig. 24. From here, it was concluded that for Poisson's equation, both partial pressure and flux must be available for the hybrid loss function to accelerate convergence. However, to train a

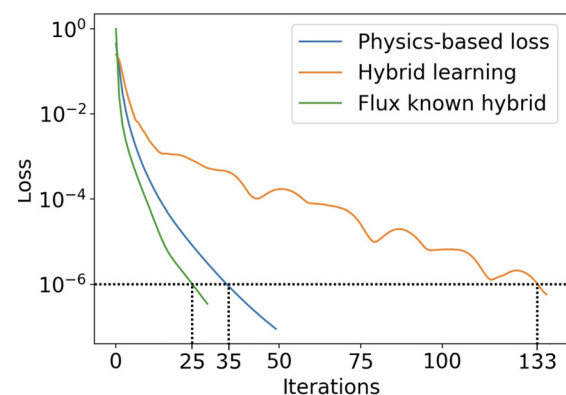


FIG. 24. Mean squared loss vs iterations. Compared with fully physics-based, the hybrid loss has delayed convergence due to non-linearly divergent flux. However, if flux is conservative and known, the hybrid loss function converges faster in 25 iterations than in 35 for fully physics-based learning.

NN to predict flux is still an open question as pressure and flux vectors are of different sizes.

Furthermore, this analysis also provides a reason for accelerated convergence in turbulence modeling by the hybrid loss functions. The reason is that eddy viscosity does not create any divergent non-linearity as it algebraically adds to the fluid viscosity. Therefore, the partially available information in turbulence can accelerate the convergence. The same is true for the energy equation as well.

REFERENCES

- ¹M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.* **378**, 686–707 (2019).
- ²X. Jin, S. Cai, H. Li, and G. E. Karniadakis, "NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations," *J. Comput. Phys.* **426**, 109951 (2021).
- ³P. Garnier, J. Viquerat, J. Rabault, A. Larcher, A. Kuhnle, and E. Hachem, "A review on deep reinforcement learning for fluid mechanics," *Comput. Fluids* **225**, 104973 (2021).
- ⁴J. Rabault and A. Kuhnle, "Accelerating deep reinforcement learning strategies of flow control through a multi-environment approach," *Phys. Fluids* **31**, 094105 (2019).
- ⁵H. Ghraieb, J. Viquerat, A. Larcher, P. Meliga, and E. Hachem, "Single-step deep reinforcement learning for open-loop control of laminar and turbulent flows," *Phys. Rev. Fluids* **6**, 053902 (2021).
- ⁶S. Pawar, S. E. Ahmed, O. San, and A. Rasheed, "Data-driven recovery of hidden physics in reduced order modeling of fluid flows," *Phys. Fluids* **32**, 036602 (2020).
- ⁷P. Pant, R. Doshi, P. Bahl, and A. Barati Farimani, "Deep learning for reduced order modelling and efficient temporal evolution of fluid simulations," *Phys. Fluids* **33**, 107101 (2021).
- ⁸S. A. Renganathan, R. Maulik, and V. Rao, "Machine learning for nonintrusive model order reduction of the parametric inviscid transonic flow past an airfoil," *Phys. Fluids* **32**, 047110 (2020).
- ⁹R. Gupta and R. Jaiman, "Three-dimensional deep learning-based reduced order model for unsteady flow dynamics with variable Reynolds number," *Phys. Fluids* **34**, 033612 (2022).
- ¹⁰L. Zhu, W. Zhang, and G. Tu, "Generalization enhancement of artificial neural network for turbulence closure by feature selection," *Adv. Aerodyn.* **4**, 1–24 (2022).
- ¹¹M. Milano and P. Koumoutsakos, "Neural network modeling for near wall turbulent flow," *J. Comput. Phys.* **182**, 1–26 (2002).
- ¹²A. Patil, J. Viquerat, A. Larcher, G. E. Haber, and E. Hachem, "Robust deep learning for emulating turbulent viscosities," *Phys. Fluids* **33**, 105118 (2021).
- ¹³L. Ma, K. Lin, D. Fan, J. Wang, and M. S. Triantafyllou, "Flexible cylinder flow-induced vibration," *Phys. Fluids* **34**, 011302 (2022).
- ¹⁴R. Gupta and R. Jaiman, "A hybrid partitioned deep learning methodology for moving interface and fluid-structure interaction," *Comput. Fluids* **233**, 105239 (2022).
- ¹⁵J. Viquerat, J. Rabault, A. Kuhnle, H. Ghraieb, A. Larcher, and E. Hachem, "Direct shape optimization through deep reinforcement learning," *J. Comput. Phys.* **428**, 110080 (2021).
- ¹⁶F.-J. Granados-Ortiz and J. Ortega-Casanova, "Machine learning-aided design optimization of a mechanical micromixer," *Phys. Fluids* **33**, 063604 (2021).
- ¹⁷Y. Wang, J. Joseph, T. A. Unni, S. Yamakawa, A. Barati Farimani, and K. Shimada, "Three-dimensional ship hull encoding and optimization via deep neural networks," *J. Mech. Des.* **144**, 101701 (2022).
- ¹⁸A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," *J. Mach. Learn. Res.* **18**, 1–43 (2018).
- ¹⁹V. Dwivedi, N. Parashar, and B. Srinivasan, "Distributed physics informed neural network for data-efficient solution to partial differential equations," *arXiv:1907.08967* (2019).
- ²⁰A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis, "Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems," *Comput. Methods Appl. Mech. Eng.* **365**, 113028 (2020).
- ²¹E. Kharazmi, Z. Zhang, and G. E. Karniadakis, "hp-VPINNs: Variational physics-informed neural networks with domain decomposition," *Comput. Methods Appl. Mech. Eng.* **374**, 113547 (2021).
- ²²V. Dwivedi and B. Srinivasan, "Physics informed extreme learning machine (PIELM)—A rapid method for the numerical solution of partial differential equations," *Neurocomputing* **391**, 96–118 (2020).
- ²³K. Li, K. Tang, T. Wu, and Q. Liao, "D3M: A deep domain decomposition method for partial differential equations," *IEEE Access* **8**, 5283–5294 (2019).
- ²⁴Z. L. Jin, Y. Liu, and L. J. Durlofsky, "Deep-learning-based surrogate model for reservoir simulation with time-varying well controls," *J. Pet. Sci. Eng.* **192**, 107273 (2020).
- ²⁵N. Wang, H. Chang, and D. Zhang, "Efficient uncertainty quantification for dynamic subsurface flow with surrogate by theory-guided neural network," *Comput. Methods Appl. Mech. Eng.* **373**, 113492 (2021).
- ²⁶Y. Zhu and N. Zabaras, "Bayesian deep convolutional encoder-decoder networks for surrogate modeling and uncertainty quantification," *J. Comput. Phys.* **366**, 415–447 (2018).
- ²⁷N. Geneva and N. Zabaras, "Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks," *J. Comput. Phys.* **403**, 109056 (2020).
- ²⁸L. Sun, H. Gao, S. Pan, and J.-X. Wang, "Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data," *Comput. Methods Appl. Mech. Eng.* **361**, 112732 (2020).
- ²⁹Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris, "Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data," *J. Comput. Phys.* **394**, 56–81 (2019).
- ³⁰J. Chen, J. Viquerat, F. Heymes, and E. Hachem, "A twin-decoder structure for incompressible laminar flow reconstruction with uncertainty estimation around 2d obstacles," *Neural Comput. Appl.* **34**, 6289–6217 (2022).
- ³¹S. Heinz, "Minimal error partially resolving simulation methods for turbulent flows: A dynamic machine learning approach," *Phys. Fluids* **34**, 051705 (2022).
- ³²C. Górlé, S. Zeoli, M. Emory, J. Larsson, and G. Iaccarino, "Epistemic uncertainty quantification for Reynolds-averaged Navier-Stokes modeling of separated flows over streamlined surfaces," *Phys. Fluids* **31**, 035101 (2019).
- ³³J. Ling, A. Kurzawski, and J. Templeton, "Reynolds averaged turbulence modeling using deep neural networks with embedded invariance," *J. Fluid Mech.* **807**, 155–166 (2016).
- ³⁴C. Rao, H. Sun, and Y. Liu, "Physics-informed deep learning for incompressible laminar flows," *Theor. Appl. Mech. Lett.* **10**, 207–212 (2020).
- ³⁵J.-T. Hsieh, S. Zhao, S. Eismann, L. Mirabella, and S. Ermon, "Learning neural PDE solvers with convergence guarantees," *arXiv:1906.01200* (2019).
- ³⁶R. Ranade, C. Hill, and J. Pathak, "DiscretizationNet: A machine-learning based solver for Navier-Stokes equations using finite volume discretization," *Comput. Methods Appl. Mech. Eng.* **378**, 113722 (2021).
- ³⁷S. Patankar and D. Spalding, "A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows," *Int. J. Heat Mass Transfer* **15**, 1787–1806 (1972).
- ³⁸H. Jasak, A. Jemcov, Z. Tukovic *et al.*, "OpenFOAM: A C++ library for complex physics simulations," in *Proceedings of the International Workshop on Coupled Methods in Numerical Dynamics* (IUC, Dubrovnik, Croatia, 2007), Vol. 1000, pp. 1–20.
- ³⁹R. Maulik, H. Sharma, S. Patel, B. Lusch, and E. Jennings, "Deploying deep learning in OpenFOAM with tensorflow," AIAA Paper No. 2021-1485, 2021, p. 1485.
- ⁴⁰J. H. Ferziger, M. Perić, and R. L. Street, *Computational Methods for Fluid Dynamics* (Springer, 2002), Vol. 3.
- ⁴¹V. Sekar, Q. Jiang, C. Shu, and B. C. Khoo, "Fast flow field prediction over airfoils using deep learning approach," *Phys. Fluids* **31**, 057103 (2019).
- ⁴²M. Xu, S. Song, X. Sun, and W. Zhang, "A convolutional strategy on unstructured mesh for the adjoint vector modeling," *Phys. Fluids* **33**, 036115 (2021).

- ⁴³J. Chen, E. Hachem, and J. Viquerat, “Graph neural networks for laminar flow prediction around random two-dimensional shapes,” *Phys. Fluids* **33**, 123607 (2021).
- ⁴⁴F. Ogoke, K. Meidani, A. Hashemi, and A. B. Farimani, “Graph convolutional networks applied to unstructured flow field data,” *Mach. Learn.: Sci. Technol.* **2**, 045020 (2021).
- ⁴⁵Y. Liu, W. Cao, W. Zhang, and Z. Xia, “Analysis on numerical stability and convergence of Reynolds averaged Navier–Stokes simulations from the perspective of coupling modes,” *Phys. Fluids* **34**, 015120 (2022).
- ⁴⁶H. Jasak, “Error analysis and estimation for the finite volume method with applications to fluid flows,” Ph.D. thesis (University of London, Imperial College London, 1996).
- ⁴⁷D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” [arXiv:1412.6980](https://arxiv.org/abs/1412.6980) (2014).
- ⁴⁸U. Ghia, K. N. Ghia, and C. Shin, “High-Re solutions for incompressible flow using the Navier–Stokes equations and a multigrid method,” *J. Comput. Phys.* **48**, 387–411 (1982).
- ⁴⁹P. Spalart and S. Allmaras, “A one-equation turbulence model for aerodynamic flows,” AIAA Paper No. 1992-439, 1992, p. 439.
- ⁵⁰V. Yakhot, S. Orszag, S. Thangam, T. Gatski, and C. Speziale, “Development of turbulence models for shear flows by a double expansion technique,” *Phys. Fluids A* **4**, 1510–1520 (1992).
- ⁵¹F. R. Menter, “Two-equation eddy-viscosity turbulence models for engineering applications,” *AIAA J.* **32**, 1598–1605 (1994).
- ⁵²J.-L. Wu, H. Xiao, and E. Paterson, “Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework,” *Phys. Rev. Fluids* **3**, 074602 (2018).
- ⁵³J.-X. Wang, J.-L. Wu, and H. Xiao, “Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on DNS data,” *Phys. Rev. Fluids* **2**, 034603 (2017).
- ⁵⁴C. Sotgiu, B. Weigand, K. Semmler, and P. Wellinger, “Towards a general data-driven explicit algebraic Reynolds stress prediction framework,” *Int. J. Heat Fluid Flow* **79**, 108454 (2019).
- ⁵⁵R. Maulik, H. Sharma, S. Patel, B. Lusch, and E. Jennings, “A turbulent eddy-viscosity surrogate modeling framework for Reynolds-averaged Navier–Stokes simulations,” *Comput. Fluids* **227**, 104777 (2021).
- ⁵⁶See https://turbmodels.larc.nasa.gov/backstep_val_sa.html for Turbulence Modeling Resource: 2D backward facing step, Spalart–Allmaras closure model; accessed 16 October 2021.
- ⁵⁷D. W. Stephens, A. Jemcov, and C. Sideroff, “Verification and validation of the caelus library: Incompressible turbulence models,” in *Proceedings of the Fluids Engineering Division Summer Meeting* (American Society of Mechanical Engineers, 2017), Vol. 58059, p. V01BT11A010.
- ⁵⁸L. Zhu, W. Zhang, J. Kou, and Y. Liu, “Machine learning methods for turbulence modeling in subsonic flows around airfoils,” *Phys. Fluids* **31**, 015105 (2019).
- ⁵⁹X. Sun, W. Cao, Y. Liu, L. Zhu, and W. Zhang, “High Reynolds number airfoil turbulence modeling method based on machine learning technique,” *Comput. Fluids* **236**, 105298 (2022).