Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 378 (2021) 113722
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

DiscretizationNet: A machine-learning based solver for Navier–Stokes equations using finite volume discretization

Rishikesh Ranade[a,*], Chris Hill[b,*], Jay Pathak[a]

[a] *Ansys Inc., Canonsburg, PA, USA*
[b] *Ansys Inc., Lebanon, NH, USA*

Received 16 May 2020; received in revised form 26 July 2020; accepted 2 February 2021
Available online 26 February 2021

Abstract

Over the last few decades, existing Partial Differential Equation (PDE) solvers have demonstrated a tremendous success in solving complex, non-linear PDEs. Although accurate, these PDE solvers are computationally costly. With the advances in Machine Learning (ML) technologies, there has been a significant increase in the research of using ML to solve PDEs. The goal of this work is to develop an ML-based PDE solver, that couples' important characteristics of existing PDE solvers with ML technologies. The two solver characteristics that have been adopted in this work are: (1) the use of discretization-based schemes to approximate spatio-temporal partial derivatives and (2) the use of iterative algorithms to solve linearized PDEs in their discrete form. In the presence of highly non-linear, coupled PDE solutions, these strategies can be very important in achieving good accuracy, better stability and faster convergence. Our ML-solver, DiscretizationNet, employs a generative CNN-based encoder–decoder model with PDE variables as both input and output features. During training, the discretization schemes are implemented inside the computational graph to enable faster GPU computation of PDE residuals, which are used to update network weights that result into converged solutions. A novel iterative capability is implemented during the network training to improve the stability and convergence of the ML-solver. The ML-Solver is demonstrated to solve the steady, incompressible Navier–Stokes equations in 3-D for several cases such as, lid-driven cavity, flow past a cylinder and conjugate heat transfer.

Keywords: Partial Differential Equations; Machine Learning; Discretization Methods; Physics-Informed Learning

1. Introduction

The coupling of physics and deep learning to solve problems in the engineering simulation space has drawn interest in recent years. In the context of neural networks, this has been achieved by constraining the network optimization by embedding physical constraints in the loss formulation. These physics-based constraints ensure that the solution space is bounded and obeys physical laws. Although this idea was proposed back in the 90s [1,2], it has started to have a big impact in recent times due to rapid advances in computational sciences and deep learning.

Within the context of physics-based deep learning there are two types of methods used to approximate the partial differential equations (PDEs) governing physical processes: data-driven and data-free. The data-driven methods use

* Corresponding authors.
E-mail addresses: rishikesh.ranade@ansys.com (R. Ranade), chris.hill@ansys.com (C. Hill), jay.pathak@ansys.com (J. Pathak).

https://doi.org/10.1016/j.cma.2021.113722
0045-7825/

simulation or experimental data to construct models while enforcing physical laws. These models heavily depend on the fidelity of the data and hence are limited in accuracy and generalizability. On the other hand, data-free methods use neural networks to generate solutions by rigorously constraining the partial differential governing equations through the loss formulation. In this respect, Raissi et al. [3,4] recently introduced the Physics Informed Neural Network (PINN) framework which approximates the partial derivatives of the solution variables with respect to space and time using automatic differentiation (AD) [5]. The partial derivatives are used to estimate the PDE losses which are back-propagated to the neural network for weight updates. In addition to the constraints on the PDE residual, the boundary and initial conditions are specified on the computational domain and constrained in the loss formulation to ensure that a unique solution is obtained at convergence. Based on this work, the PINN framework has been extended in different directions to improve the effectiveness of learning partial differential equations (PDEs) and to develop a theoretical understanding of PINNs. Recently, Kharzami et al. [6] developed the Variational PINNs to solve the PDE in its weak form. Similarly, Khodayi-Mehr et al. [7] proposed the VarNet, where the loss formulation was based on the integral form of the PDE as opposed to the differential form. In recent efforts a distributed version of PINN has been explored, where the learning problem is decomposed into smaller regions of the computational domain and a physical compatibility condition is enforced in between neighboring domains [8–11]. Other variations of PINN as well as the convergence and stability of PINN-based algorithms has been studied in [12–18].

The PINN methodology has been demonstrated to solve a number of canonical PDEs as well as to different applications involving vastly different physics [19–34]. A number of these studies have used the PINN framework to solve more complex PDEs such as the Navier–Stokes equations, which is the focus of this work. Dwivedi et al. [35] developed a Distributed-PINN to address some of the issues with PINN and demonstrated it to solve Navier–Stokes equation in a lid-driven cavity at low Reynolds numbers. Sun et al. [36] demonstrated the PINN methodology for surrogate modeling of fluid flow at low Reynolds numbers. Zhu et al. [37] implemented physical constraints on an encoder–decoder network in conjunction with flow based conditional generative models for stochastic modeling of fluid flows. Rao et al. [38] demonstrated the PINN approach to solve Navier–Stokes equations at low Reynolds numbers for a two-dimensional flow over a cylinder. Very recently, Jin et al. [39] proposed the PINN approach for solving Navier–Stokes equations in both laminar and turbulent regimes. Gao et al. [40] proposed the PhysGeoNet to solve the Navier–Stokes equations using the finite difference discretizations of the PDE residuals in the neural network loss formulation. The methodology was demonstrated on irregular domains using elliptic coordinate mappings to transform spatial coordinates.

The computation of the PDE loss and the choice of network architecture used for network optimization become crucial when solving for highly non-linear, multi-dimensional, stiff, coupled PDEs such as the system of Navier–Stokes equations. The highly non-linear solution space accessed by the Navier–Stokes equations may be challenging to resolve due to the presence of sharp local gradients in a broad computational domain. As a result, the methodology used for computing gradients as well as the approach of network training can be very important in achieving accurate solutions, better stability and faster convergence in the training process.

Traditional PDE solver technologies developed over the last few decades have primarily relied on solving discretized formulations of PDEs using methods such as, finite volume, finite element or finite difference. The exact or approximate forms of the linearized discrete equations are used, in combination with linear equation solvers, to improve the solutions iteratively. The discretization method allows access to higher order and advanced numerical approximations for partial derivatives which can be useful in resolving highly non-linear parts of the PDE solution and also add artificial dissipation to improve solver stability. Additionally, these schemes coupled with the iterative solution strategy have proved to be robust in terms of solver stability and convergence. Recently, machine-learning based models have been developed to either learn new discretization schemes from solution data [41,42] or to mimic these schemes through novelties in neural network architectures [43,44]. On the other hand, the Ansys suite of software already has access to a large number of advanced discretization schemes that can capture complexities over a wide range of physics. The coupling of these discretization schemes with machine-learning algorithms, along with the iterative solution algorithm, can provide the same benefits in ML-based solvers as observed with traditional solvers.

The main goal of this work is to introduce a new ML-solver, DiscretizationNet, which is a framework that couples solver characteristics with generative networks to solve highly non-linear, multi-dimensional, stiff, coupled PDEs. The solver does not require any training data but generates PDE solutions and simultaneously learns them during

the training process. The different finite-volume based numerical schemes are implemented inside the computational graph to enable fast, vectorized operations on GPU and a modified encoder–decoder network architecture is proposed to solve the PDEs in an iterative manner. Finally, the discretization based iterative ML solver is used to solve the steady, incompressible, Navier–Stokes equation in 3-D for a several cases such as, lid-driven cavity flow at a high Reynolds number, flow past a cylinder in laminar regime and conjugate heat transfer. The remainder of the paper is organized as follows. In Section 2, we will introduce the solution methodology adopted in the DiscretizationNet. Subsequently, Section 3 will discuss the numerical results followed with conclusions and future work in Section 4.

2. Solution methodology

In this section, we will discuss the methodology for solving the system of steady, incompressible Navier–Stokes equations using the DiscretizationNet. The system of Navier–Stokes equations consists of the continuity equation and momentum equation for each directional velocity component. The non-dimensional equations described in a vectorized form are given as follows:

$$\left.\begin{aligned} &\textbf{Continuity Equation}: && \nabla.\mathbf{v} = 0 \\ &\textbf{Momentum Equation}: && (\mathbf{v}.\nabla)\mathbf{v} + \nabla\mathbf{p} - \frac{1}{Re}\nabla^2\mathbf{v} = 0 \end{aligned}\right\} \tag{1}$$

where $\mathbf{v}$ is the non-dimensional velocity vector, $v = (u, v, w)$, $\mathbf{p}$ is the non-dimensional pressure, ∇ is the divergence operator, and Re is the Reynolds number.

When solving using neural networks, the PDEs described in Eq. (1) are used to compute the residuals in the loss formulation. Some of the reasons that can result in a stiff formulation of the PDE residual-based loss term include, complicated geometries, strong coupling of PDE variables in large system of coupled PDEs, presence of multiple domains with different PDE formulations and material properties (such as conjugate heat transfer between fluid and solid domains) and reasonably large Reynolds numbers, where the non-linear, convective component, $(\mathbf{v}.\nabla)\mathbf{v}$, is dominant. Automatic differentiation (AD) [5] allows computation of partial derivatives within the computational graph using back-propagation, but the stiffness of computed gradients may affect the accuracy, stability and convergence of the neural network training, and may require a large number of training epochs, the use of regularization techniques, as well as deep network architectures, which have their own set of problem, e.g. vanishing gradients. Considering these challenges, it becomes increasingly important to resolve such PDEs using advanced numerical schemes and develop novel strategies of neural network training. Existing solver methodologies have solved some of these problems and in this work we draw from the vast pool of knowledge to develop our ML-based solver. Next, we introduce the network architecture used in the work followed by the loss formulation and training mechanics.

2.1. DiscretizationNet architecture

The network proposed in this work is a generative Convolutional Neural Network (CNN) based encoder–decoder whose input features are flow variables, $v = (u, v, w, p)$, initialized with random uniform solution fields, boundary condition encoding, b, and the level set of the geometry, h. Level sets are real valued functions which depict the geometry such that, regions inside the geometry are flagged with -1, the region outside it with 1 and the regions representing the surface as 0 [45]. The objective of this network, as shown in, Fig. 1, is to compress the input features ($\underline{u}$, $\underline{v}$, $\underline{w}$, $\underline{p}$, $\underline{b}$, $\underline{h}$) into a lower dimensional space, (η), using a convolutional encoder and to decode the latent vector encoding to new solutions, ($\hat{u}$, $\hat{v}$, $\hat{w}$, $\hat{p}$), which are closer to actual Navier–Stokes solutions. It may be observed that the boundary conditions as well as the geometry level sets are used to enrich the encoded latent space and also in the computation of the coupled PDE loss terms, (R_c, R_u, R_v, R_w), corresponding to continuity and momentum equations in each spatial direction. The enrichment of the latent space with geometry and boundary information is crucial as it ensures that the network outputs are conditioned upon them. The geometry and boundary encoder networks are pre-trained on similar samples and their weights are used to perform encoding in this network. Encoding geometry and boundary is crucial in this approach, because, in their original form, these representations can be very sparse. This may have an adverse effect on the learning and generalizability of the DiscretizationNet. Additionally, the network proposed here can be used to compute a large batch of solutions at different boundary and geometry conditions in a single training session. As a result, the conditioning of solutions with boundary and

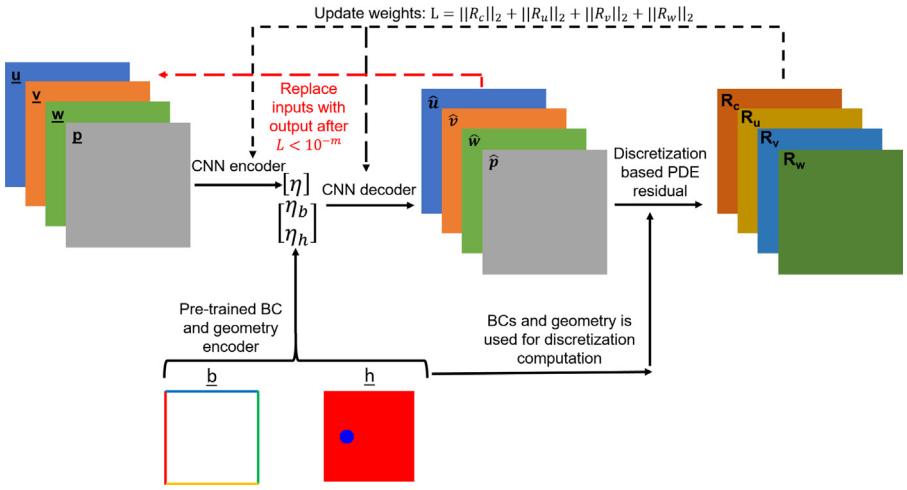

Fig. 1. DiscretizationNet for Navier–Stokes solution.

geometry conditions enables generalization of PDE solutions for a large set of problems. The purpose of designing the DiscretizationNet as an encoder–decoder network is to obtain a legitimate lower-dimensional encoding of the PDE solution space. The encoded solution space can be useful in developing reduced-order models on top of the DiscretizationNet.

2.2. Training mechanics

It was suggested previously that the input vectors to the encoder–decoder network are randomly initialized solution fields of velocity and pressure. This has two implications, (1) the PDE solution encoding (η) does not have a physical meaning and (2) decoding solutions, which are functions of random noise, is a difficult task and may slow down convergence of network and provide poor stability. In order to tackle these challenges, an iterative approach is followed, where the inputs to the network are replaced with the newly generated solutions, every time the PDE residuals reduce by an order of magnitude. At any given point during the training, the solutions generated are dependent on the solutions from a previous iteration and not the initial random solutions used in the beginning. This allows the network to converge from partially converged solutions to fully converged solutions in an iterative fashion and improves stability and convergence as compared to other ML methods in this space. At convergence, when the L-2 norm of PDE residuals have dropped to a reasonably low level, the input and output solutions are very similar and effectively turns the network into a conditional autoencoder. It is a conditional autoencoder, because the decoder network is conditioned on the solution encoding as well as the encoding of geometry and boundary. This iterative strategy provides a physical meaning to the reduced dimensional solution latent space, which can now describe flow solutions at given boundary, geometry or flow conditions. Moreover, the PDE solutions are independent of the spatial dimension and depend only on the solutions at previous iterations. This is analogous to how traditional solvers function and additionally provides an opportunity to operate this network under transient conditions. It is important to note that the training is completely data free and the goal here is to generate the solutions by minimizing the PDE residuals and simultaneously learn them into an encoded latent space.

2.3. Geometry and boundary encoder

In this section, we elaborate on the geometry and boundary encoders used in the network architecture in Fig. 1. In this work, we use a modified level set approach to represent the geometry, such that the voxels inside the geometry are represented by *0* and outside by *1*. The gradient of the level set are used to track the voxels activated by the surfaces of the geometry. The level sets for primitive geometries of different shape, size and orientation are learned using a generative encoder–decoder network and represented in a lower-dimensional space, η_h. A schematic of the

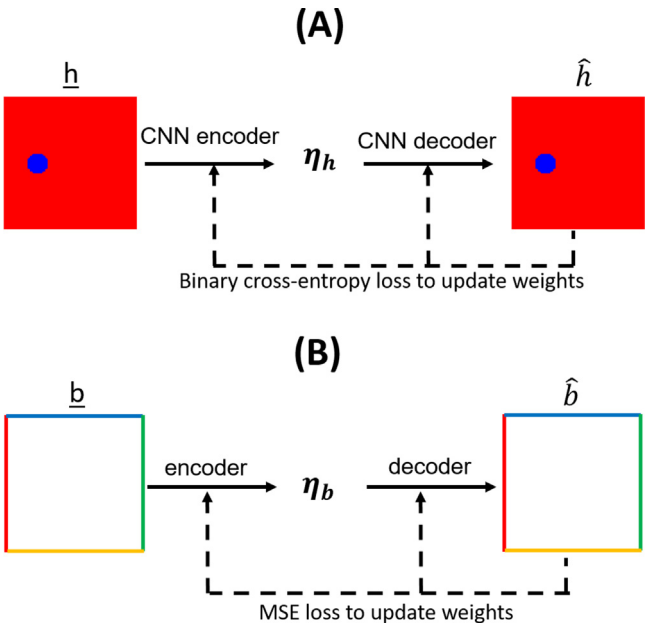

Fig. 2. Geometry and boundary autoencoders.

geometry autoencoder can be seen in Fig. 2A. In this work, the encoder and decoder networks are CNN-based and a binary cross-entropy loss function is employed to update the weights of the networks. Level sets of different geometries can be generated by parsing through the latent space vector of a trained geometry encoder and used to parameterize the ML-solver.

On the other hand, a separate boundary autoencoder is used to represent different boundary conditions. The boundary encoder is only required if the boundary conditions are spatially or temporally varying. Here, we propose a generative encoder–decoder network to learn the boundary condition encoding but leave the choice of the network architecture open. In scenarios where the boundary condition is constant along the different surfaces, as is in all of the test cases demonstrated in this work, a neural network based boundary condition encoder is not required. Instead, a custom encoding can be constructed and used as the latent vector, η_b. Flow conditions, such as Reynolds number or Prandtl number can also be perceived as boundary conditions and be added to this latent vector. For example, an encoding of $\eta_b = [1, 1, 2, 1, 0.3, 1.2, 0, 3, 40]$ can be perceived as Dirichlet inlet boundary conditions (1) on left, right and bottom surfaces with specified values of 0.3, 1.2 and 3.0, respectively, and a Neumann boundary condition (2) on the top surface, where the flux of variable equal zero. The Reynolds number (40) or other flow conditions can also be specified in the encoding. A similar choice of boundary condition encoding is employed in this work.

2.4. Loss formulation

The loss formulation of the network comprises of PDE residual from all the governing equations. Each PDE has its own loss formulation given as follows:

$$\lambda(\boldsymbol{W}, \boldsymbol{b}) = \|\lambda_c\|_\Omega + \|\lambda_u\|_\Omega + \|\lambda_v\|_\Omega + \|\lambda_w\|_\Omega \tag{2}$$

where $\|\lambda_c\|_\Omega$ is the L-2 norm of the continuity residual, $\|\lambda_u\|_\Omega$ is the L-2 norm of the x-momentum residual, $\|\lambda_v\|_\Omega$ is the L-2 norm of the y-momentum residual, $\|\lambda_w\|_\Omega$ is the L-2 norm of the z-momentum residual and Ω is solution space. Moreover, this network can generate and learn a large set of solutions at different Reynolds numbers. The different solutions simply form a part of the training samples of the encoder–decoder network.

The computation of PDE residuals involves approximation of first and second order spatial gradients. As mentioned earlier, we employ the traditional finite-volume discretization technique to compute the PDE residual loss

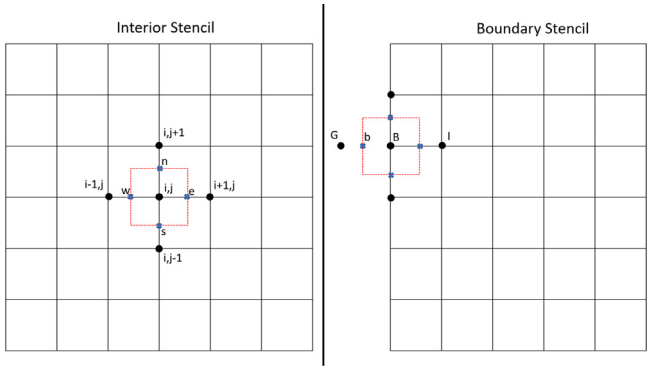

Fig. 3. Finite-volume stencil on the interior and boundary pixels in a computational graph.

and moreover, all the numerical schemes are implemented inside the computational graph to enable fast, vectorized GPU computation. However, this does not limit the use of other discretization schemes such as Finite Element Method (FEM), Discontinuous Galerkin (DG) etc., if higher order elements are required.

In the finite-volume discretization implemented here, each voxel of the PDE solution is considered as the cell center of an imaginary, finite control volume (CV) as shown in Fig. 3. The volume integrals on the CV are expressed as surface integrals using the Green–Gauss divergence theorem shown below.

$$\int (\nabla . v)\, \mathrm{d}V = \sum_{j}^{m} v_j n_j A_j \tag{3}$$

where, v is a solution variable, M is the number of faces on the CV, n_j is the normal along each face on CV and A_j is the area of each face on CV. Hence, a face-based approach is used to compute gradients across all interior and boundary faces of each control volume in the computation graph. The second order gradients are also computed using Eq. (3), with the only difference that the solution variable is replaced by its gradient. The convective fluxes are discretized using the first or second-order upwind scheme and the diffusive fluxes are evaluated using a central difference approximation. A second order approximation is used for computing gradients of the pressure field. Since we are dealing with an incompressible formulation of the Navier–Stokes and the discretization is analogous to finite volume discretization on collocated grids, the pressure field is prone to checker-boarding due to a lack of explicit coupling between pressure and velocity and the use of second order numerical schemes. In this work, the pressure–velocity coupling is achieved using the Rhie–Chow interpolation [46], which essentially adds a fourth order dissipation of pressure to the continuity equation. The addition of Rhie–Chow flux suffices and more sophisticated schemes such as SIMPLE [47] are not required but remain an option. In the Rhie–Chow formulation, the velocity at the faces is interpolated as shown in Eq. (4):

$$u_e = \left(\frac{u_i + u_{i+1}}{2}\right) + \frac{1}{a_p}\left(p_i + \nabla P_i . \bar{r}_i - p_{i+1} - \nabla P_{i+1} \bar{r}_{i+1}\right) \tag{4}$$

where, u_e is the velocity approximation at the *east* face of the control volume, p is the pressure, ∇P is the gradient of pressure and a_p are the matrix coefficients from the momentum equations.

The boundary condition treatment is incorporated through the discretization of boundary voxels by enforcing boundary constraints in the flux computation and enforces the order of accuracy at the boundary, as opposed to using one-sided finite difference schemes. For example, the x-velocity gradients along a boundary as shown in Fig. 3 is represented as follows:

$$\left.\begin{aligned} \left(\frac{\mathrm{d}u}{\mathrm{d}x}\right)_G &= 0.5\left(\frac{u_B + u_G}{\Delta x_B}\right) - 0.5\left(\frac{u_B + u_I}{\Delta x_I}\right) \\ u_G &= 2u_b - u_B \end{aligned}\right\} \tag{5}$$

where, u_b is the specified boundary condition, u_G is an imaginary ghost pixel, u_B is the boundary voxel and u_I is the interior voxel adjacent to the boundary in the direction of the gradient. In the case of unstructured boundaries,

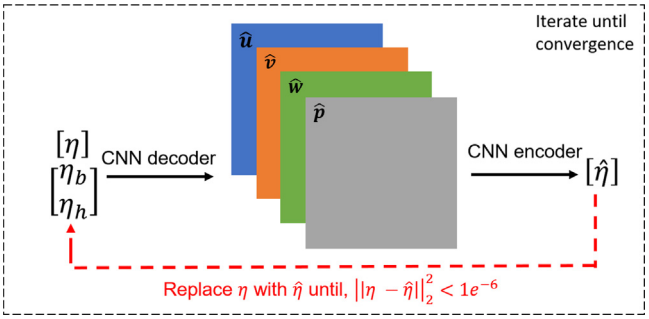

Fig. 4. Algorithm for inferencing.

such as, cells adjacent to the walls of a cylinder, a stair-step discretization [48] or a cut-cell discretization [49] can be implemented. In this work, we have implemented the stair-step discretization, where the boundary conditions at unstructured boundaries are implemented similarly as in Eq. (5).

The numerical schemes at both interior and boundary voxels are implemented together in the computational graph using vectorized operation for fast computation on GPU. The discretization schemes are implemented through custom hidden layers in Keras [50], where the solution variable tensors as well as boundary and geometry condition tensors are used to compute the PDE residuals of the coupled PDE system at each voxel. As a result, loss formulation in Eq. (2) implicitly contains boundary information and a separate loss term is not needed to model it, as in previous studies [3,4], thereby avoiding the need to use strategies for multi-objective optimization. Moreover, the use of discretization techniques to compute loss, allows access to higher order approximations for higher accuracy and advanced numerical schemes such as Rhie–Chow flux [46] etc. that can enhance stability and convergence of solution and neural network training by providing additional physics-based regularization.

2.5. Inference for other geometry and boundary conditions

The network architecture described in Fig. 1 is a generative encoder–decoder network, where, at convergence, the input and output samples are essentially the actual solutions of given PDEs. It may be understood that the model obtained at convergence has learned to encode actual PDE solutions and decode them from the solution latent space combined with geometry and boundary encoding. As a result, the model in its current form cannot be directly used for inferencing solutions at other geometry and boundary conditions, since actual PDE solutions may be required as inputs to the network and these are obviously not available.

Here, we propose a novel algorithm that enables inferencing for other geometry and boundary conditions. A schematic diagram of the algorithm is shown in Fig. 4 and the important steps are outlined below.

1. On a given geometry and boundary condition initialized for inference, the geometry and boundary encoding, η_h and η_b are computed using their respective encoder networks.
2. Since, the solution encoding, η, is unknown, it is initialized with a random field drawn from a uniform distribution.
3. The initial solution encoding combined with the geometry and boundary encoding is passed through the trained weights of the CNN decoder to generate a solution field, $\hat{u}$, $\hat{v}$, $\hat{w}$, $\hat{p}$.
4. The solution field is encoded to a new solution encoding using the trained weights of the CNN encoder, $\hat{\eta}$.
5. The new solution encoding, $\hat{\eta}$, replaces the solution encoding of the previous iteration, η and steps (iii) and (iv) are repeated until the L-2 norm of $\eta - \hat{\eta} < 1e^{-6}$. The geometry and boundary encoding are fixed during the entire process.
6. At convergence, the PDE solutions at a given geometry and boundary condition are decoded using the most recent η.

It may be observed that the solution inference happens in the encoded latent vector space and the goal of the iterative procedure is to steer the solution latent vector to a space that is in close proximity to the latent vectors

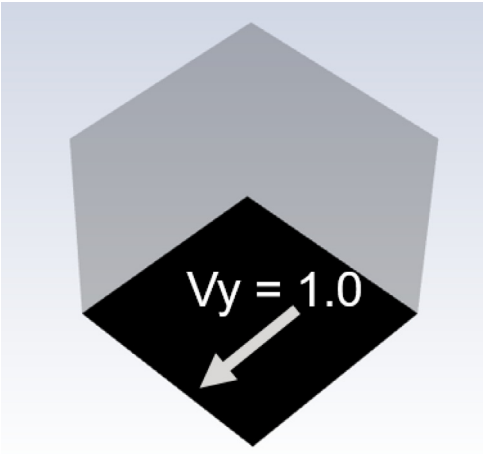

Fig. 5. Description of computational domain for lid-driven cavity flow.

observed in the network training. Since the geometry and boundary condition encoding are fixed for a given problem, they provide the necessary constraints for the solution latent vector to iteratively improve itself and generate an accurate PDE solution. The outcome of this algorithm, in terms of generalization, improves with the number of different variations of geometry and boundary conditions adopted during training. Generally, starting from the weights obtained from a well trained model, the inference algorithm converges in fewer than 10 iterations. Although not a scope of our current work, functioning in the space of latent vectors may provide an opportunity to explore new solution spaces of a given PDE and construct computationally inexpensive reduced order models.

3. Results

This section provides detailed numerical experiments to demonstrate the ML-solver for several cases of fluid flow such as lid-driven cavity, flow past a cylinder and conjugate heat transfer. The proposed ML-solver is validated against the ANSYS Fluent 19.3 CFD [51] solver for solving the incompressible, steady Navier–Stokes equation for these different cases.

3.1. Lid-driven cavity flow

In this experiment, the 3-D lid-driven cavity problem is solved at three different Reynolds numbers (Re), 1000, 2500 and 5000. In 3-D, this problem is known to become unsteady for Re numbers above 3000 [52] and may cause convergence issues in CFD solvers for steady, incompressible Navier–Stokes equation. Such issues are not experienced with the ML-Solver, which is able to solve up to Reynolds numbers of 10,000 and even higher, due to inherent regularization in the ML techniques. As a result, the Reynolds numbers of 1000 and 2500 are chosen to obtain fair comparisons with traditional CFD solvers, since they fall in the steady regime and yet exhibit substantial non-linearity. Streamline plots at Reynolds number of 5000 are presented to demonstrate the ability of the ML-solver to operate at higher Reynolds numbers due to its inherent regularizing capability. Moreover, it is important to note that these solutions are different training samples to a single neural network and hence, they are all learned and generated simultaneously.

A schematic diagram of the problem is shown in Fig. 5. The boundary conditions are applied such that the bottom wall is moving in the y-direction with a specified velocity. The boundary velocity and domain dimensions are constant and the kinematic viscosity is tuned with respect to the test Reynolds numbers. The computational domain chosen for this example has a resolution of 64^3. Although, it is important to mention that the resolution of the domain has no bearing on the accuracy or performance of this method with respect to traditional CFD flow solvers. Finer or coarser resolutions can be used depending on the problem definition and the solution fidelity required. The implementation of the ML algorithm is carried out in Keras [50], which provides a lot of flexibility in creating custom hidden layers that are useful in developing the discretization schemes inside computation graphs,

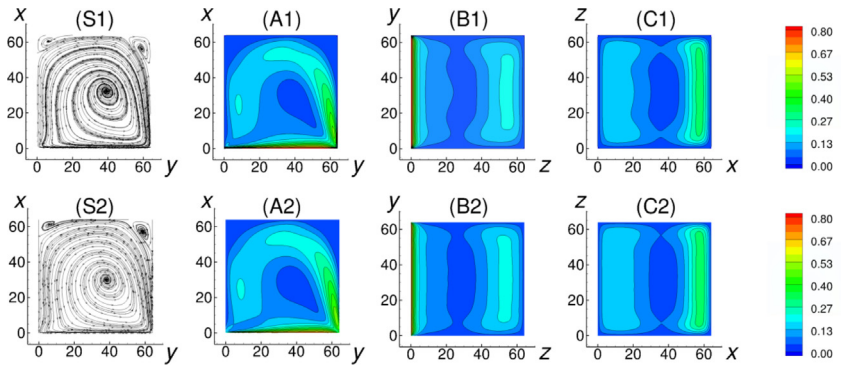

Fig. 6. Comparison of velocity magnitude in a Lid-driven cavity at Re = 1000 between ANSYS Fluent(1) and ML-Solver(2) on planes (A) x-y, (B) x-z and (C) y-z cut along the center of the domain.

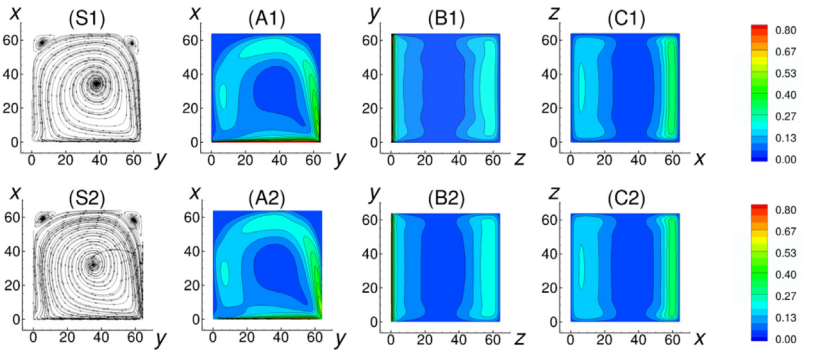

Fig. 7. Comparison of velocity magnitude in a Lid-driven cavity at Re = 2500 between ANSYS Fluent(1) and ML-Solver(2) on planes (A) x-y, (B) x-z and (C) y-z cut along the center of the domain.

as discussed before. The CNN encoder and decoder for all solution variables have 3 layers each with 64 filters in each layer and tangent–sigmoid activation function. A maxpooling layer is used in the encoder network, in between the 3 layers and conversely, an upsampling layer is used in the decoder network. Additionally, the different solution variables are intertwined in the bottleneck layer through the use of fully connected dense layers. The choice of network architecture and the associated hyper-parameters may be changed depending on the complexity of problem at hand. In this work, the hyper-parameters of the network are chosen with a big factor of safety and may not be optimal. Hence, the same network architecture works reasonably well for the 3 test cases in consideration but is not general by any means. A sensitivity analysis can be conducted to determine the optimum network architecture and other hyper-parameters. The training is carried out using an Adam optimizer with an initial learning rate of $1e^{-3}$. A learning rate scheduler is used to tune the learning rate depending on the network loss. Since, we are using an iterative approach where the input samples are replaced with the output, the learning rate is reinitialized to $1e^{-3}$, every time the iterative operation is implemented. For all test cases, about $3\text{x}10^4$ training iterations are carried out on a NVIDIA Tesla V100 SXM2 GPU.

The predicted velocity magnitude for both Reynolds numbers are shown in Figs. 6 and 7. The results from the ML-Solver are compared with results generated from ANSYS Fluent R19.3 [51]. The discretization schemes used are consistent in both cases. The comparisons are carried out on planes cut through the center of the domain along all three dimensions (A, B and C) as well as a stream line plot along y-z plane through the center of the domain (S). The contour plots depict the velocity magnitude while the streamlines are plotted for the velocity in the y-direction. It may be observed from contour plots that the predictions of the ML-solver match reasonably well with those from ANSYS Fluent. The streamline plots show that the global flow features such as the positions of the primary and the secondary vortical structures are captured well.

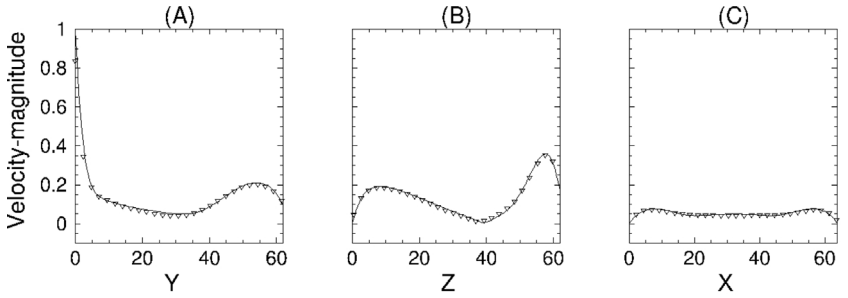

Fig. 8. Comparison of velocity magnitude in a Lid-driven cavity at Re = 1000 between ANSYS Fluent(symbol) and ML-Solver(solid) on centerline along (A) y, (B) z and (C) x.

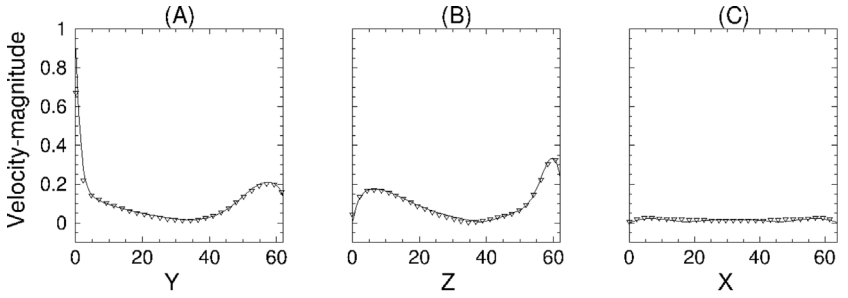

Fig. 9. Comparison of velocity magnitude in a Lid-driven cavity at Re = 2500 between ANSYS Fluent(symbol) and ML-Solver(solid) on centerline along (A) y, (B) z and (C) x.

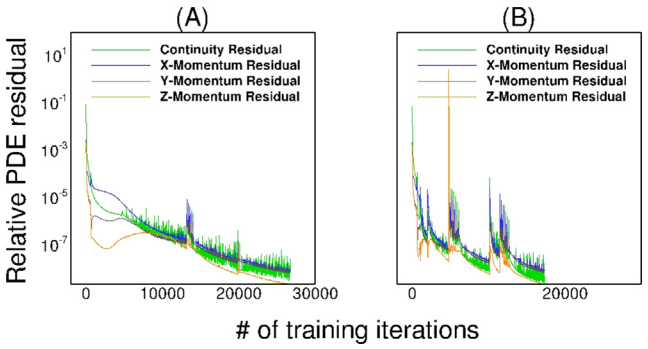

Fig. 10. Comparison of residual convergence, (A) without using iterative strategy and (B) using iterative strategy. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Although, contour and streamline plots provide a good qualitative comparison of the flow structure, the line plots along the centerline of domain provide a more quantitative comparison between the two solutions. In Figs. 8 and 9, the velocity magnitude is plotted on lines passing through the center of the domain along x, y and z directions. It may be observed that the results agree well and the relative errors are less than 1% at each point, attributing to the good accuracy of the ML-solver. Fig. 10 shows a comparison of the convergence of relative residuals (with respect to initial residuals) of continuity, x-momentum, y-momentum and z-momentum equations for training with and without the iterative procedure. It may be observed that the convergence of solutions to a relative residual of $1e^{-8}$ is faster when the iterative procedure is implemented. Spurious peaks may be observed in the convergence plots with iterative procedure. Since the network weights are tuned for a given input vector, the replacement of inputs with outputs causes the PDE residuals to momentarily jump. It may be observed that this behavior is only observed in the beginning, when the output solutions are significantly different from the input vectors. As the PDE residuals reduce and the solutions get closer to convergence, the inputs and outputs have fewer differences and any

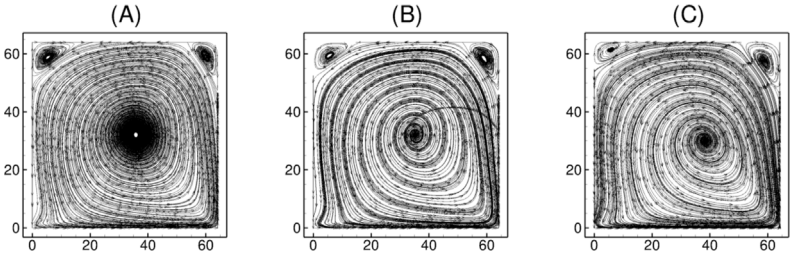

Fig. 11. Streamline plot comparisons on the center x-y in a Lid-driven cavity at Re $=$ (A) 5000, (B) 2500 and (C) 1000.

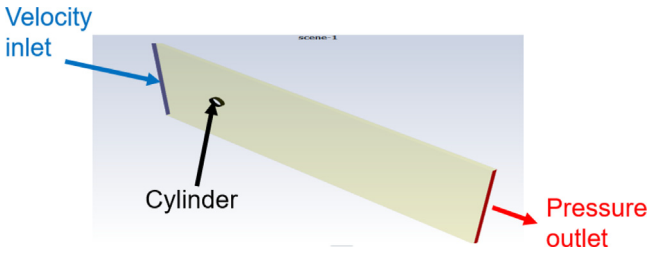

Fig. 12. Description of computational domain for flow past a cylinder.

replacements made after this point results in a smoother convergence. Additionally, it is important to note that in both cases, the total number of training epochs are less than $3\text{x}10^4$ and each training epoch requires a computation time of about 1 s. The computation of discretization in the computation graph does not add significantly to the training cost due to the vectorized implementation, and in fact improves training stability, thus resulting in fast convergence.

Next, we present a comparison of streamline plots obtained from the ML-Solver at $Re = 5000$ in Fig. 11 with other Reynolds numbers. The streamlines are plotted along plane cut through the center of the domain along z direction. The ML-solver provides convergent solutions for the steady, incompressible Navier–Stokes equation at $Re = 5000$ and even beyond, as a result of the added physics-based regularization, but solutions from other PDE solvers do not converge easily for Reynolds numbers beyond 3000, hence those comparisons are not provided. It may be observed from Fig. 11 that an increase in Reynolds number results in clear differences in the size and position of the primary and secondary vortical structures, attributing to the ability of the ML-Solver to capture solutions at higher Reynolds numbers.

The validation of the solver for the lid-driven cavity case at reasonably high Reynolds numbers shows that the ML-solver can generate accurate non-linear flow solutions as well as result in better stability and fast training in $3-$dimensional scenarios. With that being said, lid-driven cavity is a relatively simpler case where the geometry is not complicated, there is minimal interaction between fluid and solid domains, and the only source of stiffness in the PDEs results from the non-linearity in the solution field. In the following sections, we validate the ML-solver for cases involving complicated geometries and non-trivial fluid–solid interactions.

3.2. Laminar flow past a cylinder

In this section, we validate the performance of the ML-solver in solving the 3-D, steady, incompressible Navier–Stokes equations for flow past a cylinder at 3 different Reynolds numbers in the steady regime, $Re = 10, 20, 40$. A schematic diagram of the problem is shown in Fig. 12. The computational domain extends to about $30D$ in the x-direction and $10D$ in the y and z directions, where D is the diameter of the cylinder. The computational mesh is highly resolved with 320 elements in $x-$ direction, 128 elements in $y-$ direction and 4 elements in $z-$ direction. Since the solver employs a stair-step discretization, a reasonably high mesh resolution is required to effectively represent the curvature of the cylinder. A coarser or finer resolution may be used depending on the size of the cylinder and the flow Reynolds number. The mesh resolution does not have any bearing on the accuracy and

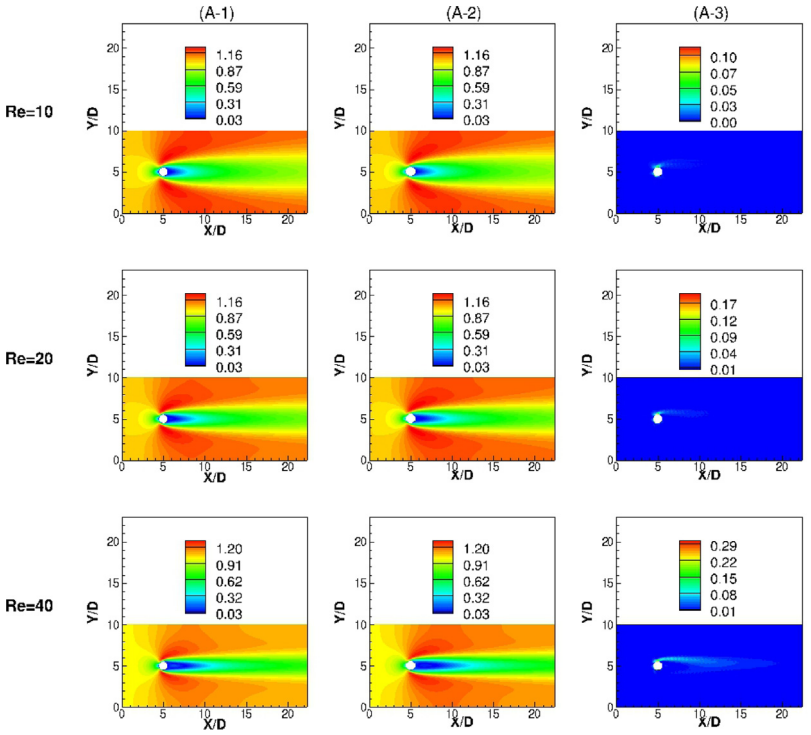

Fig. 13. Comparison of velocity magnitude at different Reynolds numbers between ML-Solver (1), Ansys Fluent (2) and mean squared error (3) for an inlet velocity of 1.0 (A).

performance of results generated by the ML-Solver with respect to other flow solvers. The left boundary of the computational domain is specified as the velocity inlet, while the right boundary is the pressure outlet. All the other boundaries, perpendicular to the cylinder, are specified as symmetric. The training is carried out using the same network architecture and procedure as described in the previous section. The network is trained to generate solutions at the previously stated Reynolds numbers as well as 5 different velocity inlet conditions for each Reynolds number given as, 0.2, 0.4, 0.6, 0.8, 1.0.

The predicted velocity magnitude as well as pressure for a solution at a velocity inlet boundary condition of 1.0, generated during training, are presented in Figs. 13 and 14 for the three Reynolds numbers. It may be observed that the solutions are in great agreement with the solutions obtained from Ansys Fluent 19.3 [51], the relative errors are small and the flow structures on the upstream and downstream of the cylinder are captured well. Most of the errors are observed near the cylinder walls.

Next, we compare line plots of pressure and x-velocity at two different locations, $x/d = 6$ and center, $y/d = 5$ for the same case, as in Figs. 13 and 14. The line plot comparisons, presented in Fig. 15, show that the solutions obtained from the ML-Solver are less than 5% of solutions obtained from Ansys Fluent [51]. The differences in the solutions can be attributed to the discretization error near the curved surfaces of the cylinder. As mentioned earlier, the ML-solver employs a stair-step discretization to capture the cylinder surface and that can be dissipative, if the mesh is not fine enough. On the other hand, Ansys Fluent uses an unstructured grid discretization, which provides a better representation of the cylinder surface and thus, results in slightly more accurate solutions, as can be observed from the line as well as contour plots. This issue can be circumvented by adding a cut-cell discretization capability [49] or an unstructured grid discretization in the ML-Solver and couple it with graph CNNs [53] or Mesh-based CNNs [54] to perform convolution and pooling operations on unstructured domains.

Further, to demonstrate the ability of the model to predict at velocity conditions that were not observed during training, we use the inferencing algorithm, described previously in Fig. 4, to evaluate the solution at a test inlet velocity condition of 0.5 for all three Reynolds numbers. The inlet velocity used for testing was not generated or learned during network training. The main purpose of this exercise is to demonstrate the adequacy and validate the

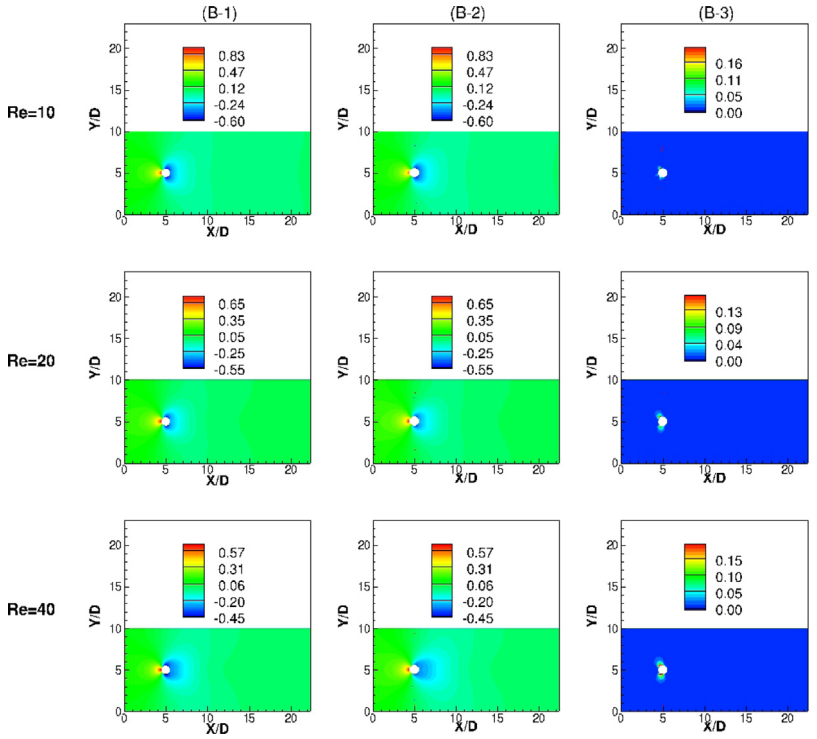

Fig. 14. Comparison of pressure at different Reynolds numbers between ML-Solver (1), Ansys Fluent (2) and mean squared error (3) for an inlet velocity of 1.0 (B).

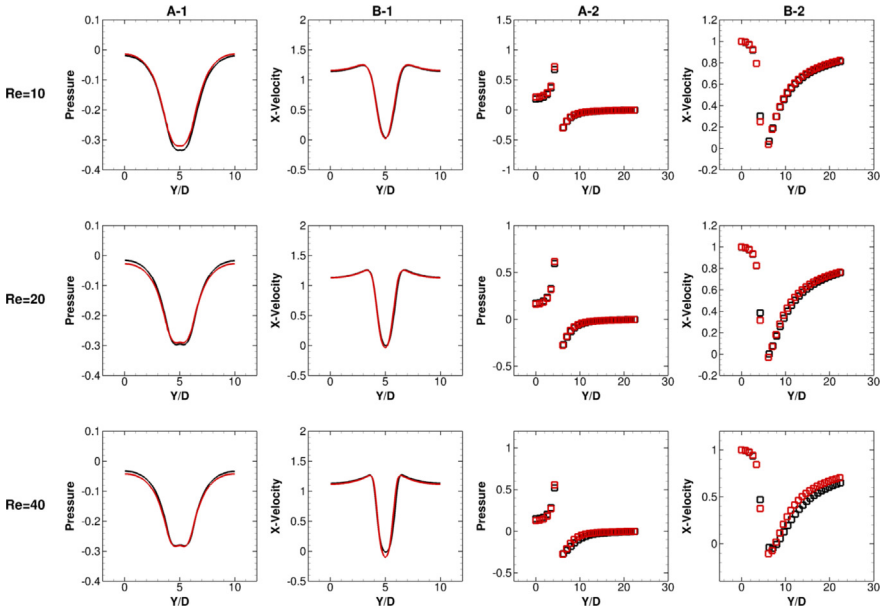

Fig. 15. Comparison between ML-solver (red) and Ansys Fluent (black) for x-velocity (A) and pressure (B) at different Reynolds numbers plotted at, (1) $x/d = 6$ and (2) center line at inlet velocity of 1.0. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

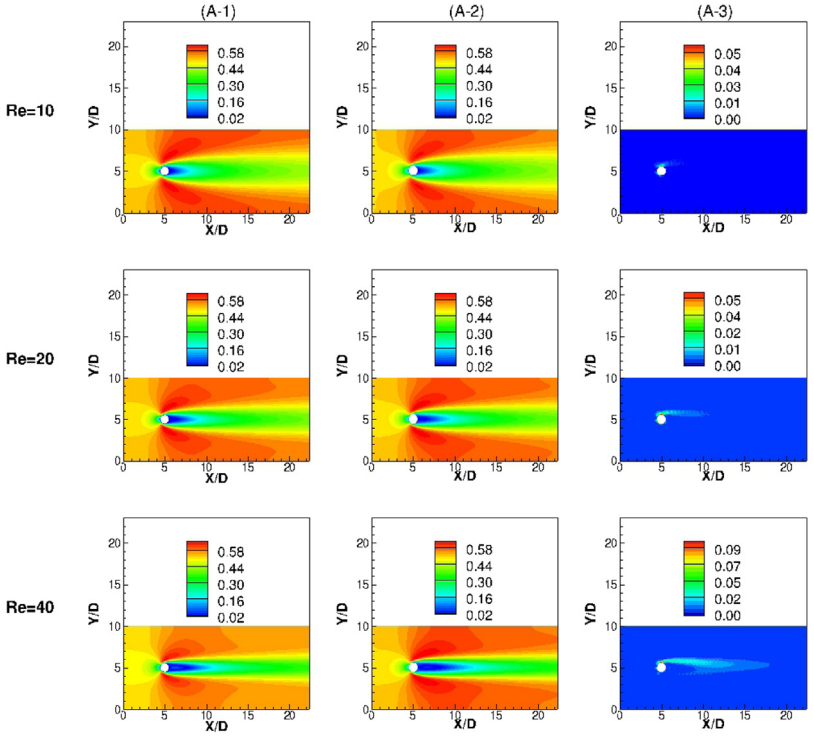

Fig. 16. Comparison of velocity magnitude (A) at different Reynolds numbers between ML-Solver (1), Ansys Fluent (2) and mean squared error (3) for an inlet velocity of 0.5.

inferencing algorithm. The predictions of velocity magnitude and pressure at the test velocity are shown in Figs. 16 and 17. It may be observed that the predictions from the inferencing algorithm match reasonably well with the solutions obtained from Ansys Fluent [51].

3.3. Conjugate heat transfer

Finally, we demonstrate the ML-solver to solve a conjugate heat transfer problem in a laminar flow setting ($Re = 10$) and compare the results with Ansys Fluent [51]. A schematic of the problem description in provided in Fig. 18. The computational domain consists of two parts, a fluid and a solid domain. The fluid domain extends 64 m in each direction with velocity and temperature inlet specified on the left boundary and pressure outlet on the right. The computational mesh contains 64 elements in each direction. The mesh near the geometry is relatively coarser than the one used for flow past a cylinder. All the other surfaces are specified as symmetry. On the other hand, the solid domain is a smaller cube, and is placed inside the fluid domain. Seven different side lengths of solid cubes ranging from, 8 m and 20 m, are considered during training. The thermal diffusivity in the solid is assumed to be 2 times as that of the fluid. A Gaussian heat source, depicted in Fig. 18, is described at the center of the solid domain as shown in Eq. (6).

$$P\left(\bar{x}\right) = 0.05 * \left(\frac{1}{\sqrt{(2\pi)^3 det(\Sigma)}} exp\left(-\frac{1}{2}(\bar{x} - \mu)^T \Sigma^{-1} (\bar{x} - \mu) \right) \right) \tag{6}$$

where, μ is the mean and Σ is the covariance matrix, where the variance in each direction equal to 0.15.

The network architecture and procedure used during training is similar to the cases described previously. The only difference lies in the loss formulation, where different equations are solved in the fluid and solid domains. In the fluid domain, we solve the Navier–Stokes equation, Eq. (1), and Energy equation, Eq. (7), while the Heat Conduction equation is solved in the solid domain, Eq. (7). A one-way coupling is used at the interface between

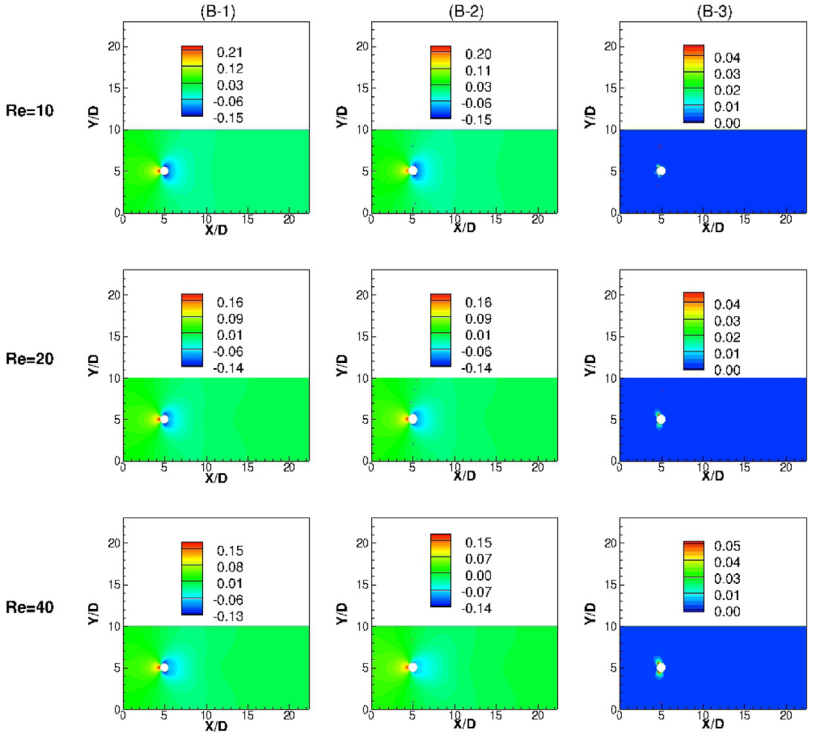

Fig. 17. Comparison of pressure (B) at different Reynolds numbers between ML-Solver (1), Ansys Fluent (2) and mean squared error (3) for an inlet velocity of 0.5.

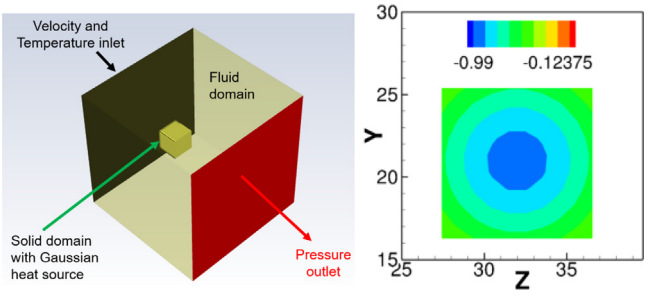

Fig. 18. Description of computational domain for conjugate heat transfer (left) and a contour of Gaussian power map on a center-plane along Y–Z (right).

the fluid and solid domains. The Energy and Heat equations are shown in Eq. (7).

$$\left.\begin{aligned} &\textbf{Heat Equation}: && \nabla.(\alpha\nabla\mathbf{T}) - P = 0 \\ &\textbf{Energy Equation}: && (\mathbf{v}.\nabla)\mathbf{T} - \nabla.(\alpha\nabla\mathbf{T}) = 0 \end{aligned}\right\} \tag{7}$$

where $\mathbf{v}$ is the velocity vector, $v = (u, v, w)$, $\mathbf{T}$ is the normalized temperature based on inlet temperature, ∇ is the divergence operator, P is the heat source, and α is the thermal diffusivity. The Navier–Stokes equation and energy equation discretization's are employed in the fluid domain while the Heat equation discretization with power source is used in the solid domain. The solution for an additional PDE for energy as well as the contrasting properties and the coupling between the solid and fluid domains results in additional challenges for the ML-solver as compared to the case of flow past a cylinder, presented previously.

Next, we compare the predicted normalized temperature and velocity profile generated by the ML-solver during training with solutions from Ansys Fluent [51] for a 10 m long solid cube. The contour plots presented in Fig. 19

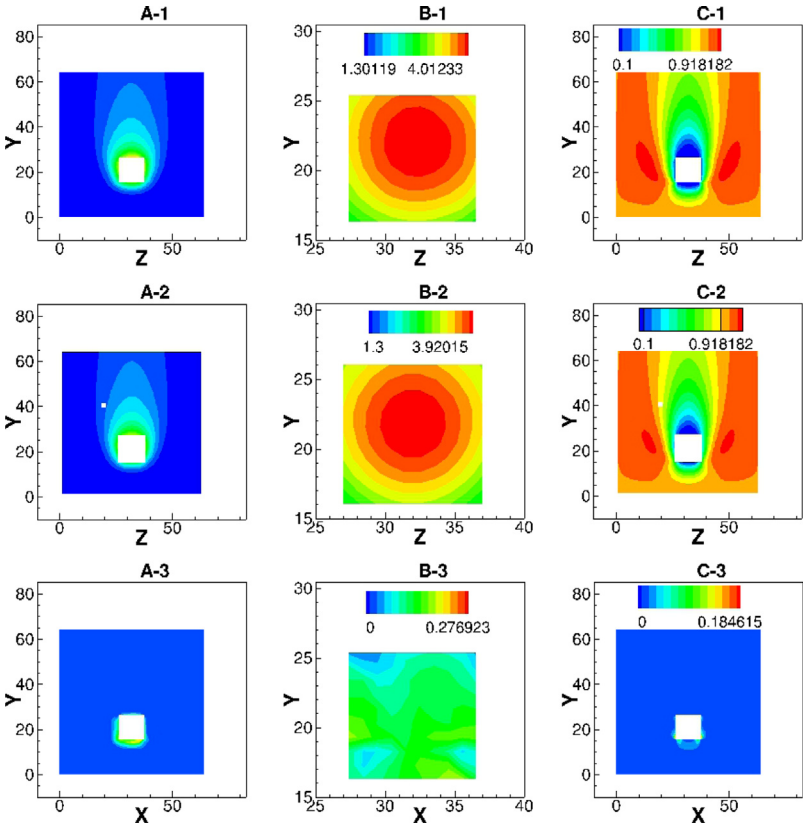

Fig. 19. Comparison of normalized temperature in fluid domain (A), solid domain (B) and y-velocity (C) between ML-solver (1), Ansys Fluent (2) and mean squared error between them (3) for a solid square cube of size 10 m.

show the temperature profile in the fluid and the solid domain, as well as the velocity-magnitude profile in the fluid domain. It may be observed that the peak temperature occurs inside the solid domain due to the presence of a heat source and the temperature from the solid dissipates into the fluid. The direction of dissipation and the magnitude of peak temperature are affected by the velocity field and the flow Reynolds number. The ML-solver is able to predict the physics reasonably well in comparison to Ansys Fluent. The error plots in Fig. 19 show that the solutions from the ML-solver are in close agreement with the Ansys Fluent 19.3 [51] solutions in both fluid and solid domains. Although lower in magnitude, most of the errors occur near the walls and can be attributed to the stair-step discretization employed by the ML-Solver. The near-wall error in this case is larger than the flow past cylinder case because the near-wall mesh resolution is coarser in this case. Nonetheless, the results are in reasonable agreement.

Fig. 20 shows line plot comparisons of normalized temperature and y-velocity between the ML-solver and Ansys Fluent [51] at $y = 21$ and $y = 40$. The ML-solver results are within 1% of the Fluent solutions in both fluid and solid domains, thus validating the accuracy of the ML-solver.

Next, we use the inferencing algorithm described in Fig. 4 to predict the temperature and velocity fields for a square cube of size, 13 m, which was not a part of the training set. It may be observed from the contour plots in Fig. 21 that the predictions of the ML-solver match well with those from Ansys Fluent [51]. Additionally, it may also be observed that the peak normalized temperature in the solid domain is lower in this case as compared to when the size of the solid cube is smaller, attributing to the difference in their surfaces areas, thereby validating the underlying physics that a larger surface area results in an increased heat loss from the solid into the fluid domain. The line plot comparisons are shown in Fig. 22. It may be observed that the there is great agreement in the predictions of ML-solver as compared to Ansys Fluent [51] and that the relative error in the predictions is less than 1%.

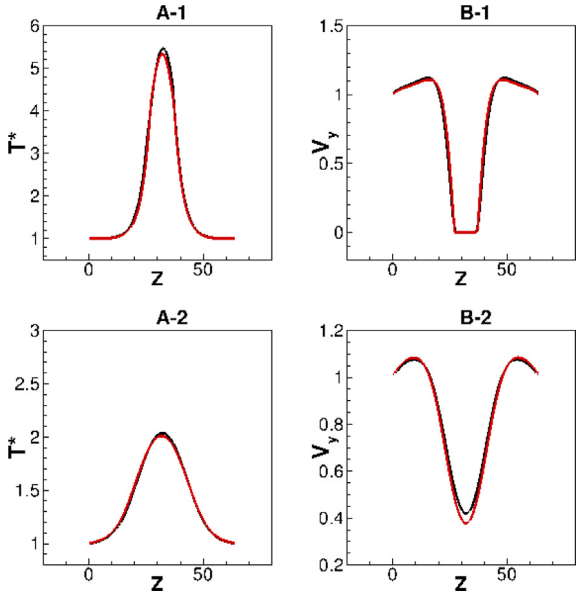

Fig. 20. Comparison of normalized temperature (A) and y-velocity (B) between ML-solver (red) and Ansys Fluent (black) at $y = 21$ (1) and $y = 40$ (2) for a solid square cube of size 10 m. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

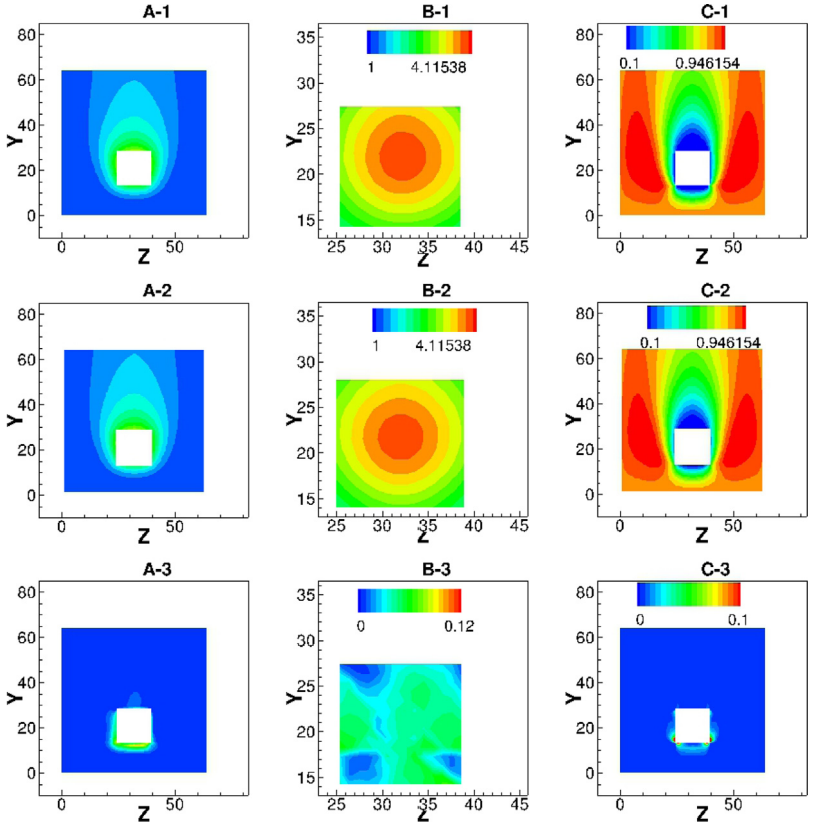

Fig. 21. Comparison of normalized temperature in fluid domain (A), solid domain (B) and y-velocity (C) between ML-solver (1), Ansys Fluent (2) and mean squared error between them (3) for a solid square cube of size 13 m.

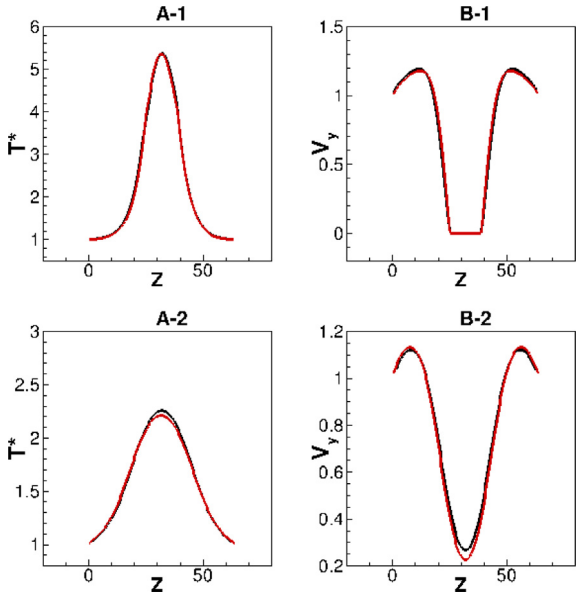

Fig. 22. Comparison of normalized temperature (A) and y-velocity (B) between ML-solver (red) and Ansys Fluent (black) at $y = 21$ (1) and $y = 40$ (2) for a solid square cube of size 13 m. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4. Conclusion

In this work, we have presented a novel ML-Solver, which uses important characteristics from existing PDE solvers for solving the system of steady, incompressible Navier–Stokes equation. The ML-solver does not require any training data and instead, generates and learns the PDE solutions simultaneously, during the training process. It uses discretization techniques to approximate the PDE residual at each voxel of a given computational domain and uses the L-2 norm of the residual to update network weights. The discretization schemes are implemented inside the computational graph to enable vectorization on GPU and provide access to numerous higher order and advanced numerical schemes that can enhance the accuracy as well as improve stability of the ML-solver, through physics-based regularization. In this work, we have extended the discretizations to unstructured domains by employing stair-step discretizations to provide flexibility in modeling different types of geometries as well as widely varying boundary conditions.

From the network architecture perspective, we introduce the DiscretizationNet, which is a generative CNN-based encoder–decoder network conditioned on geometry and boundary conditions. Separate autoencoders are constructed to learn lower-dimensional vectors (or encodings) for different geometry and boundary conditions. These encodings are used to enrich and parameterize the solution latent vector space of the generative network and thus allow for simultaneously generating and learning a wide range of solutions at different conditions in the same training session. Moreover, we employ a novel iterative capability in the network to mimic existing PDE solvers. In this implementation, the inputs to the generative model are replaced with outputs during network training, as the network learns to generate better solutions. This strategy is unique and we have observed that it provides better stability and faster convergence in comparison to other ML strategies, especially in cases when the ground truth solutions are not known. Additionally, we have proposed an algorithm for inferencing using the DiscretizationNet. The algorithm functions in the latent space to iteratively infer solutions using the trained model weights.

We have validated the ML-solver by solving the 3-D steady, incompressible Navier–Stokes equations on three different cases, (i) lid-driven cavity, (ii) laminar flow past a cylinder and (iii) conjugate heat transfer. Contour and line plot comparisons made with ANSYS Fluent R19.3 [51] in all three cases show a good agreement. Additionally, it has been observed that the training for a large number of PDE solutions results in a stable convergence within $3\text{x}10^4$ training epochs.

The ML-solver proposed here can be extended to solve unsteady problems using LSTM-type networks [55]. The deficiencies in stair-step discretization, in computing accurate solutions near the boundaries can be mitigated by

using a cut-cell of unstructured grid discretization. Moreover, the ML-solver can be applied to other PDEs with complex physics as well as to develop computationally inexpensive, low-dimensional models.

Declaration of competing interest

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

References

[1] H. Lee, I.S. Kang, Neural algorithm for solving differential equations, J. Comput. Phys. 91 (1) (1990) 110–131.
[2] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000.
[3] M. Raissi, G.E. Karniadakis, Hidden physics models: Machine learning of nonlinear partial differential equations, J. Comput. Phys. 357 (2018) 125–141.
[4] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, 2017, arXiv preprint arXiv:1711.10561.
[5] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. 18 (1) (2017) 5595–5637.
[6] E. Kharazmi, Z. Zhang, G.E. Karniadakis, Variational physics-informed neural networks for solving partial differential equations, 2019, arXiv preprint arXiv:1912.00873.
[7] R. Khodayi-Mehr, M.M. Zavlanos, Varnet: Variational neural networks for the solution of partial differential equations, 2019, arXiv preprint arXiv:1912.07443.
[8] K. Li, K. Tang, T. Wu, Q. Liao, D3m: A deep domain decomposition method for partial differential equations, IEEE Access 8 (2019) 5283–5294.
[9] E. Kharazmi, Z. Zhang, G.E. Karniadakis, Hp-VPINNs: Variational physics-informed neural networks with domain decomposition, 2020, arXiv preprint arXiv:2003.05385.
[10] V. Dwivedi, B. Srinivasan, Physics informed extreme learning machine (PIELM)–a rapid method for the numerical solution of partial differential equations, Neurocomputing 391 (2020) 96–118.
[11] A.D. Jagtap, E. Kharazmi, G.E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems, Comput. Methods Appl. Mech. Engrg. 365 (2020) 113028.
[12] M. D'Elia, G.E. Karniadakis, G. Pang, M.L. Parks, Nonlocal physics-informed neural networks-a unified theoretical and computational framework for nonlocal models., in: AAAI Spring Symposium: MLPS, 2020.
[13] E. Haghighat, A.C. Bekar, E. Madenci, R. Juanes, A nonlocal physics-informed deep learning framework using the peridynamic differential operator, 2020, arXiv preprint arXiv:2006.00446.
[14] Y. Zang, G. Bao, X. Ye, H. Zhou, Weak adversarial networks for high-dimensional partial differential equations, J. Comput. Phys. (2020) 109409.
[15] X. Meng, Z. Li, D. Zhang, G.E. Karniadakis, PPINN: Parareal physics-informed neural network for time-dependent PDEs, Comput. Methods Appl. Mech. Engrg. 370 (2020) 113250.
[16] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient pathologies in physics-informed neural networks, 2020, arXiv preprint arXiv:2001.04536.
[17] Y. Shin, J. Darbon, G.E. Karniadakis, On the convergence and generalization of physics informed neural networks, 2020, arXiv preprint arXiv:2004.01806.
[18] L. Yang, D. Zhang, G.E. Karniadakis, Physics-informed generative adversarial networks for stochastic differential equations, 2018, arXiv preprint arXiv:1811.02033.
[19] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, DeepXDE: A deep learning library for solving differential equations, 2019, arXiv preprint arXiv:1907.04502.
[20] J. Berg, K. Nyström, A unified deep artificial neural network approach to partial differential equations in complex geometries, Neurocomputing 317 (2018) 28–41.
[21] M. Raissi, Z. Wang, M.S. Triantafyllou, G.E. Karniadakis, Deep learning of vortex-induced vibrations, J. Fluid Mech. 861 (2019) 119–137.
[22] M. Raissi, H. Babaee, P. Givi, Deep learning of turbulent scalar mixing, Phys. Rev. Fluids 4 (12) (2019) 124501.
[23] M. Raissi, A. Yazdani, G.E. Karniadakis, Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations, Science 367 (6481) (2020) 1026–1030.
[24] G. Kissas, Y. Yang, E. Hwuang, W.R. Witschey, J.A. Detre, P. Perdikaris, Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks, Comput. Methods Appl. Mech. Engrg. 358 (2020) 112623.
[25] Z. Fang, J. Zhan, Deep physical informed neural networks for metamaterial design, IEEE Access 8 (2019) 24506–24513.
[26] D. Liu, Y. Wang, Multi-fidelity physics-constrained neural network and its application in materials modeling, J. Mech. Des. 141 (12) (2019).
[27] X. Chen, J. Duan, G.E. Karniadakis, Learning and meta-learning of stochastic advection-diffusion-reaction systems from sparse measurements, 2019, arXiv preprint arXiv:1910.09098.

[28] Y. Chen, L. Lu, G.E. Karniadakis, L. Dal Negro, Physics-informed neural networks for inverse problems in nano-optics and metamaterials, Opt. Express 28 (8) (2020) 11618–11633.
[29] Q. Zheng, L. Zeng, G.E. Karniadakis, Physics-informed semantic inpainting: Application to geostatistical modeling, J. Comput. Phys. (2020) 109676.
[30] Z. Mao, A.D. Jagtap, G.E. Karniadakis, Physics-informed neural networks for high-speed flows, Comput. Methods Appl. Mech. Engrg. 360 (2020) 112789.
[31] X. Meng, G.E. Karniadakis, A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems, J. Comput. Phys. 401 (2020) 109020.
[32] A.M. Tartakovsky, C.O. Marrero, P. Perdikaris, G.D. Tartakovsky, D. Barajas-Solano, Learning parameters and constitutive relationships with physics informed deep neural networks, 2018, arXiv preprint arXiv:1808.03398.
[33] J. Berg, K. Nyström, Data-driven discovery of PDEs in complex datasets, J. Comput. Phys. 384 (2019) 239–252.
[34] L. Yang, X. Meng, G.E. Karniadakis, B-pinns: Bayesian physics-informed neural networks for forward and inverse pde problems with noisy data, 2020, arXiv preprint arXiv:2003.06097.
[35] V. Dwivedi, N. Parashar, B. Srinivasan, Distributed physics informed neural network for data-efficient solution to partial differential equations, 2019, arXiv preprint arXiv:1907.08967.
[36] L. Sun, H. Gao, S. Pan, J.-X. Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Comput. Methods Appl. Mech. Engrg. 361 (2020) 112732.
[37] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, J. Comput. Phys. 394 (2019) 56–81.
[38] C. Rao, H. Sun, Y. Liu, Physics-informed deep learning for incompressible laminar flows, 2020, arXiv preprint arXiv:2002.10558.
[39] X. Jin, S. Cai, H. Li, G.E. Karniadakis, Nsfnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations, 2020, arXiv preprint arXiv:2003.06496.
[40] H. Gao, L. Sun, J.-X. Wang, Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parametric PDEs on irregular domain, 2020, arXiv preprint arXiv:2004.13145.
[41] J. Zhuang, D. Kochkov, Y. Bar-Sinai, M.P. Brenner, S. Hoyer, Learned discretizations for passive scalar advection in a 2-d turbulent flow, 2020, arXiv preprint arXiv:2004.05477.
[42] Y. Bar-Sinai, S. Hoyer, J. Hickey, M.P. Brenner, Data-driven discretization: machine learning for coarse graining of partial differential equations, 2018, Preprint.
[43] J.-T. Hsieh, S. Zhao, S. Eismann, L. Mirabella, S. Ermon, Learning neural PDE solvers with convergence guarantees, 2019, arXiv preprint arXiv:1906.01200.
[44] B. Stevens, T. Colonius, Finitenet: A fully convolutional LSTM network architecture for time-dependent partial differential equations, 2020, arXiv preprint arXiv:2002.03014.
[45] S. Osher, J.A. Sethian, Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations, J. Comput. Phys. 79 (1) (1988) 12–49.
[46] C. Rhie, W.L. Chow, Numerical study of the turbulent flow past an airfoil with trailing edge separation, AIAA J. 21 (11) (1983) 1525–1532.
[47] S.V. Patankar, A calculation procedure for two-dimensional elliptic situations, Numer. Heat Transfer 4 (4) (1981) 409–425.
[48] J.H. Seo, R. Mittal, A sharp-interface immersed boundary method with improved mass conservation and reduced spurious pressure oscillations, J. Comput. Phys. 230 (19) (2011) 7347–7363.
[49] P. Tucker, Z. Pan, A cartesian cut cell method for incompressible viscous flow, Appl. Math. Model. 24 (8–9) (2000) 591–606.
[50] F. Chollet, et al., Keras, 2015.
[51] A. Fluent, 19.3, theory guide, ansys, 2019.
[52] A.Y. Gelfgat, Linear instability of the lid-driven flow in a cubic cavity, Theor. Comput. Fluid Dyn. 33 (1) (2019) 59–82.
[53] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, 2016, arXiv preprint arXiv:1609.02907.
[54] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, D. Cohen-Or, Meshcnn: A network with an edge, ACM Trans. Graph. 38 (4) (2019) 90.
[55] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780.