

CFDNet: A deep learning-based accelerator for fluid simulations

Octavi Obiols-Sales

oobiols@uci.edu

University of California, Irvine

Nicholas Malaya

nicholas.malaya@amd.com

Advanced Micro Devices, Inc.

Abhinav Vishnu

abhinav.vishnu@amd.com

Advanced Micro Devices, Inc.

Aparna Chandramowlishwaran

amowli@uci.edu

University of California, Irvine

ABSTRACT

CFD is widely used in physical system design and optimization, where it is used to predict engineering quantities of interest, such as the lift on a plane wing or the drag on a motor vehicle. However, many systems of interest are prohibitively expensive for design optimization, due to the expense of evaluating CFD simulations.

To render the computation tractable, reduced-order or *surrogate* models are used to accelerate simulations while respecting the convergence constraints provided by the higher-fidelity solution. This paper introduces *CFDNet* – a *physical simulation and deep learning coupled framework*, for accelerating the convergence of Reynolds Averaged Navier-Stokes simulations. CFDNet is designed to predict the primary physical properties of the fluid including velocity, pressure, and eddy viscosity using a single convolutional neural network at its core. We evaluate CFDNet on a variety of use-cases, both extrapolative and interpolative, where test geometries are observed/not-observed during training. Our results show that CFDNet meets the convergence constraints of the domain-specific physics solver while outperforming it by 1.9 – 7.4× on both steady laminar and turbulent flows. Moreover, we demonstrate the generalization capacity of CFDNet by testing its prediction on new geometries unseen during training. In this case, the approach meets the CFD convergence criterion while still providing significant speedups over traditional domain-only models.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks; Model development and analysis; Multiscale systems.**

KEYWORDS

Computational fluid dynamics, Deep learning, Physics-Machine Learning coupled framework, Turbulent flows, AI for science

ACM Reference Format:

Octavi Obiols-Sales, Abhinav Vishnu, Nicholas Malaya, and Aparna Chandramowlishwaran. 2020. CFDNet: A deep learning-based accelerator for fluid simulations. In *2020 International Conference on Supercomputing (ICS '20)*, June 29–July 2, 2020, Barcelona, Spain. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3392717.3392772>

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

ICS '20, June 29–July 2, 2020, Barcelona, Spain

© 2020 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-7983-0/20/06.

<https://doi.org/10.1145/3392717.3392772>

1 INTRODUCTION

Computational Fluid Dynamics (CFD) is the *de-facto* method for solving the Navier-Stokes equations – a challenging multi-scale multi-physics problem. Typically, the Navier-Stokes equations, a set of Partial Differential Equations (PDEs), are solved iteratively until convergence for each cell of a discretized geometry (also known as *grid* or *mesh*). These simulations are computationally expensive and there are widespread efforts to improve the performance and scalability of solving these systems [11, 18, 23, 24, 31].

The primary advancements in scaling CFD simulations have been achieved through algorithmic innovations and computational optimizations. The majority of research has focused on developing *domain-specific* optimizations. At the same time, machine learning (ML) methods – especially deep learning (DL) algorithms have demonstrated remarkable improvements in a variety of modeling/classification tasks such as computer vision, natural language processing, and high-performance computing [5, 17, 21, 34]. *Our objective in this paper is to explore the potential of DL algorithms to accelerate the convergence of CFD simulations.*

Several researchers have applied DL algorithms to accelerate CFD simulations. Table 1 summarizes the state-of-the-art (SOTA) methods by comparing them on nine different features. While the current approaches have shown the feasibility and potential of deep learning for accelerating fluid flow simulations, they suffer from fundamental limitations. First, some of the current approaches do not satisfy the *conservation laws*. In particular, DL is used to predict the output variables (such as velocity/pressure) mostly as an end-to-end *surrogate*. This does not guarantee meeting the convergence constraints of the traditional physics-based algorithms which is critical for domain scientists and engineers. Second, most of the current methods only predict a partial flow field. For example, the neural network predicts only a subset of the flow variables which provide incomplete information about the problem. Others do not

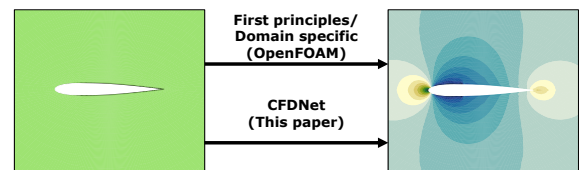


Figure 1: An example of the input and output of a CFD solver. Our objective with CFDNet is to produce the same output as the physics solver (i.e., respect the convergence constraints) while accelerating the convergence process using DL.

Table 1: Comparing CFDNet with SOTA in CFD acceleration using deep learning on nine features. CFDNet is the first deep learning and physics coupled framework to predict all relevant fluid variables in the entire fluid domain of a turbulent flow to accelerate CFD simulations while meeting the convergence constraints of the original physics solver, whereas prior work [8, 20, 36, 37] replace specific steps of the original algorithm with a deep neural network (DNN) (e.g., find a surrogate of the Poisson solver [8, 36], estimate the Reynolds stresses [20], estimate the eddy viscosity [37]) and [14, 35] replace the entire algorithm with a convolutional neural network-based surrogate to estimate the final solution of the fluid variables of interest.

Related Work	Eulerian flows	Laminar Flows	Turbulent Flows	Viscous terms	Test Geometry Unseen in Training	Predict Mean Velocity	Predict Pressure	Predict Turbulent Viscosity	Meets convergence constraints
Tompson et al. [36]	●	●	●	●	●	●	●	●	●
Smart-fluidnet [8]	●	●	●	●	●	●	●	●	●
Guo et al. [14]	●	●	●	●	●	●	●	●	●
TBNN [20]	●	●	●	●	●	●	●	●	●
Maulik et al. [22]	●	●	●	●	●	●	●	●	●
Zhu et al. [37]	●	●	●	●	●	●	●	●	●
Thuerey et al. [35]	●	●	●	●	●	●	●	●	●
CFDNet (This paper)	●	●	●	●	●	●	●	●	●

support turbulent flows that are common in most industrial applications. Third, half of the current approaches lack generalization since the test geometry is the same or a subset of the training geometry. Generalization using deep learning is challenging especially for unseen geometries. Overall, we observe that SOTA approaches satisfy up to six out of the nine features shown in Table 1.

In this paper, we tackle the above limitations and aim to develop new modeling techniques to accelerate CFD simulations, specifically Reynolds Averaged Navier-Stokes (RANS) while respecting the convergence constraints. We propose CFDNet – a deep learning framework that combines domain-specific knowledge to meet the same convergence constraints of the physical solver. An illustration is shown in Figure 1.

To address the above challenges, we must consider several choices in designing CFDNet including input/output representation (e.g., extracted/raw features, augmentation), design of the DNN (e.g., single/multiple networks for predicted variables, encoder-decoder structure, temporal vs non-temporal output), and algorithm design for combining domain-specific knowledge (e.g., warmup before executing DL algorithm and final refinement for adhering to the convergence constraints).

Contributions and Findings. We evaluate the above design choices and summarize the contributions in this paper as follows.

- We create an input-output representation consisting of multiple channels (one for each predicted flow variable) and a six-layer encoder-decoder convolutional neural network (CNN) with domain-specific activation functions to predict *all* the flow variables of interest (e.g., velocity in each direction, pressure, and eddy viscosity of the fluid). We augment the training set by using each intermediate iteration of the RANS simulation as an input while considering the output of the domain-specific solver as the target variable.
- We precondition the input to the CNN with a *warmup* method, where the first few iterations (determined automatically and adaptively based on the inference use-case) use the domain-specific physics solver before calling the CNN inference.

We show that the proposed warmup technique to capture domain knowledge reduces the overall refinement steps accelerating the final simulation by 1.9 – 4.6×

- We propose an *iterative refinement* stage, where the output from the CNN is fed back as the initial condition to the physics solver to meet the domain-specific convergence constraints. This circumvents the critical limitation in current approaches where the neural network is modeled as an end-to-end *surrogate* [14, 35] and there is no guarantee that the conservation laws are satisfied.
- We consider several use-cases to evaluate the accuracy, performance, and generalizability of CFDNet as an accelerator. These include training and prediction on the same geometry (e.g., channel flow), training on multiple geometries (ellipses) and predicting on a subset (e.g., ellipse with a different aspect ratio) which is common in DL research, and finally, training on multiple geometries (ellipses) and predicting the flow around a new geometry (e.g., airfoil or cylinder) which is challenging. CFDNet exhibits remarkable generalizability to geometries unseen during training while achieving 1.9–7.4× speedup on laminar and turbulent flows over a widely used physics solver in OpenFOAM.

The rest of the paper is organized as follows. We present the necessary CFD background in Section 2 followed by the CFDNet design in Section 3. Dataset generation and training recipe are presented in Section 4, followed by an in-depth discussion of results and analysis in Section 5. Finally, we present the related work in Section 6 and conclude in Section 7.

2 BACKGROUND

The steady incompressible RANS equations provide an approximate time-averaged solution to the incompressible Navier-Stokes equations. They describe turbulent flows as follows:

$$\frac{\partial \bar{U}_i}{\partial x_i} = 0 \quad (1)$$

$$\bar{U}_j \frac{\partial \bar{U}_i}{\partial x_j} = \frac{\partial}{\partial x_j} \left[-(\bar{p}) \delta_{ij} + (v + v_t) \left(\frac{\partial \bar{U}_i}{\partial x_j} + \frac{\partial \bar{U}_j}{\partial x_i} \right) \right] \quad (2)$$

where \bar{U} is the mean velocity (a 2D or 3D vector field), \bar{p} is the kinematic mean pressure, v is the fluid viscosity, and v_t is the eddy viscosity resulting from Boussinesq's approximation [7]. Typically, turbulence modeling is used for the eddy viscosity (v_t). The Spalart-Allmaras one-equation model shown below provides a single transport equation to compute a modified eddy viscosity, \tilde{v} .

$$\begin{aligned} \bar{U}_i \frac{\partial \tilde{v}}{\partial x_i} = & C_{b1} (1 - f_{t2}) \tilde{S} \tilde{v} - \left[C_{w1} f_w - \frac{C_{b1}}{\kappa^2} f_{t2} \right] \left(\frac{\tilde{v}^2}{d} \right) \\ & + \frac{1}{\sigma} \left[\frac{\partial}{\partial x_i} \left((v + \tilde{v}) \frac{\partial \tilde{v}}{\partial x_i} \right) + C_{b2} \frac{\partial \tilde{v}}{\partial x_j} \frac{\partial \tilde{v}}{\partial x_j} \right] \end{aligned} \quad (3)$$

Then, we can compute the eddy viscosity from \tilde{v} as $v_t = \tilde{v} f_{v1}$. These equations represent the most commonly-used implementation of the Spalart-Allmaras model. The terms f_{v1} , \tilde{S} , and f_{t2} are model-specific and contain, for instance, first order flow features (magnitude of the vorticity). C_{b1} , C_{w1} , C_{b2} , κ , and σ are constants specific to the model, calibrated experimentally. The equations and values of these constants are detailed in [33], the first original reference of the model.

Equations (1), (2), and (3) form a system of four PDEs in 2D and five PDEs in 3D. We numerically solve the discretized form of these equations on a structured grid with its corresponding boundary conditions (that define the physical boundaries). The spatially partial derivatives are numerically computed using finite difference methods. First, the gradients of the flow variables in the grid cell faces are numerically calculated with a second-order, least-squares interpolation method using the neighboring cells. Then, for the advection of the velocity and the modified eddy viscosity, we use a second-order, upwind-biased, unbounded scheme. Finally, the diffusion terms in Equation (2) and Equation (3) are evaluated using Gaussian integration, with a linear interpolation method for the viscosity calculation. Depending on the non-orthogonality level of the grid, a correction is made to ensure second-order accuracy to compute the surface normal gradient, which is required in the Laplacian calculation.

To find the steady-state solution, we set the necessary boundary conditions and numerically solve the conservation equations until convergence using the semi-implicit method for pressure-linked equations (SIMPLE) algorithm [28] outlined in Algorithm 1. It is an iterative procedure widely used in literature [19, 26, 27] for steady-state problems. In each iteration, the velocity, pressure, and eddy viscosity fields in the entire grid domain are computed (Lines 2-5) and if the convergence criterion is met in Line 6 (i.e., the residual is less than the user-defined tolerance, ϵ), these fields are considered to be final and the flow has converged to steady-state. If not, these fields are intermediate and the algorithm starts the next iteration from Line 2. Algorithm 1 is computationally expensive and the goal of this paper is to *short-circuit* the convergence progress by

Algorithm 1 SIMPLE algorithm.

```

1: Guess  $\bar{p}^*$ , an intermediate but incorrect  $\bar{p}$  field
2: Solve  $\bar{U}^*$ , an intermediate but incorrect  $\bar{U}$  field, from Equations (2)
3: Solve  $\bar{p}'$ , a correction value for  $\bar{p}$ , from Equation (1)
4: Obtain correct  $\bar{U}$  and  $\bar{p}$  fields from  $\bar{p}^*$ ,  $\bar{U}^*$ , and  $\bar{p}'$ 
5: Solve  $v_t$  from Equation (3)
6: if residual  $< \epsilon$  then
7:   return done
8: else
9:    $\bar{p}^* \leftarrow \bar{p}$ 
10:  goto 2

```

reducing the number of iterations of the RANS simulation. We will refer to Algorithm 1 as the *physics solver* for the rest of this paper.

3 CFDNET: A DL ACCELERATOR FOR FLUID SIMULATIONS

Our objective is to develop a function G that accelerates the convergence of the physics solver. Let N represent the number of iterations required for convergence with the physics solver in Algorithm 1, and I ($0 \leq I \leq N - 1$) represent any intermediate iteration of the solver. We aim to create a map such that $x^N = G(x^I; \theta_{wl})$ where $x^N, x^I \in \mathbb{R}^{m \times n \times z}$ represent the tensors of z flow variables on a $m \times n$ grid domain at convergence and any intermediate iteration respectively.

We choose CNN as the candidate for G in this paper (where θ_{wl} are the w weights in layer l that need to be calibrated) due to its ability to map complex functions – yet remain domain-agnostic. Furthermore, neural networks have demonstrated their ability to learn non-linear relationships between input and output variables. Therefore, we expect the CNN to capture the nonlinear, multi-dimensional relations present in the Navier-Stokes equations through the non-linear activation functions.

In this section, we first describe the different choices in the neural network design including input/output representation and network architecture. Then, we present the overall CFDNet framework which combines domain-knowledge with CNN inference to accelerate fluid simulations and finally, present our discussion on convergence constraints.

3.1 Input/Output Representation

We begin with a discussion of the input/output representation (i.e., how the tensors x^I and x^N are created). A 2D cartesian and incompressible RANS flow computes $z = 4$ flow variables – mean velocity's first component (\bar{u}), mean velocity's second component (\bar{v}), the mean kinematic relative pressure (\bar{p}), and the modified eddy viscosity (\bar{v}^1). At each iteration, these four flow variables are updated at every cell of the grid. Harnessing the "image" nature of the computational grid, where each cell can be interpreted as a pixel, we generate the input-output representation in Figure 2 as follows:

- (1) We compose an image of size $m \times n$, where $m \times n$ is the 2D grid size. Every pixel (i, j) of this image has the corresponding

¹The Spalart-Allmaras turbulence model uses the transport equation to solve the modified eddy viscosity, so it becomes the fourth variable in our input-output representation.

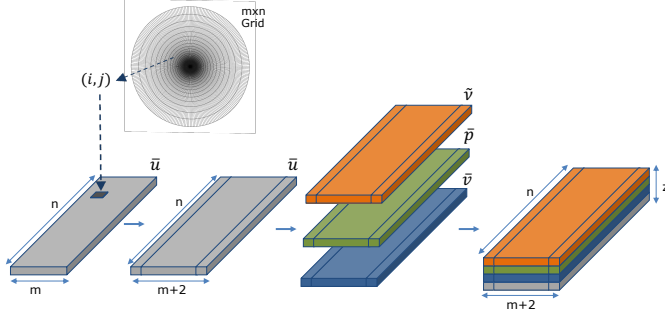


Figure 2: Input-Output Representation in CFDNet. The input and the output are images of size $(m+2) \times n \times z$ where $m \times n$ is the grid size and z is the number of flow variables (channels). The input image has the grid values of the variables at an intermediate iteration and the output image has the values of the final steady solution.

value of a flow variable, say \bar{u} in the cell (i, j) as shown in the figure. The spatial distribution of the pixels needs to respect the logical spatial arrangement of the fluid domain. Next, we augment the rows of this image by appending the physical boundary values of the variables from the computational grid (i.e., the values at the top and bottom boundaries) leading to an image size of $(m+2) \times n$.

- (2) We repeat the previous step for the other flow variables of interest – \bar{v} , \bar{p} and $\bar{\nu}$, leading to four images in total, each of size $(m+2) \times n$.
- (3) We concatenate the previously generated four images to create the final input/output representation to the CNN which is a tensor of size $(m+2) \times n \times 4$ (i.e., an image consisting of four channels).

Note that both input and output images share the same tensor dimensionality $(m+2) \times n \times z$. The only difference is that the input tensor contains fluid values of an intermediate iteration while the output tensor is the final, converged, and steady solution.

A critical step in the generation of the input/output representation is to *non-dimensionalize* the flow variables. This consists of dividing each of the variables in every cell by a flow configuration-specific reference value which is U_r for velocity, p_r for pressure, and ν_r for eddy viscosity. This is a common practice in fluid mechanics [6] to reduce the number of free parameters. We non-dimensionalize the flow variables for two key reasons both aimed at improving the learning task of the network. (i) The z flow variables that represent different physical quantities can have a vastly different range of values (for example, from 0 to 1 m s^{-1} for velocity and $1\text{e-}6$ to $1\text{e-}4 \text{ m}^2/\text{s}$ for the modified eddy viscosity). Non-dimensionalization rescales the values of the variables to the same range across all z fluid variables. (ii) It reduces the number of free parameters; if certain non-dimensional parameters are significantly smaller than others, they are negligible in certain areas of the flow [6].

3.2 Network Design and Architecture

We choose a deep neural network to learn the *surrogate* model for accelerating the RANS equations. This network is composed of

six layers, three convolution and three deconvolution layers. The output is a prediction of *all* the steady-state flow variables (\bar{u} , \bar{v} , \bar{p} , and $\bar{\nu}$) as shown in Figure 3.

The choice of a convolution-deconvolution network is motivated by two main reasons. First, recent works have successfully leveraged similar architectures for physical system emulation [9, 25]. Second, the spatial layout of the flow variables in the input tensor is a result of Equations (1), (2), and (3). The convolution operator is an optimal candidate to extract the existing spatial correlations in and among the fluid variables (i.e., input channels).

We do not include any explicit *domain-specific* feature in the network’s input and output because we aim to learn a model that generalizes to a broad number of design use cases comprising of both different geometries - interpolated and extrapolated - and flow conditions. Prior works that embed domain-knowledge in the neural network *learning* task (such as embedding Equation (1) as the loss function in [36]) fail to achieve this generalization limiting its scope.

The first three convolution layers reduce the dimensionality extracting an abstract representation of the input to encapsulate it in the latent space (middle layer colored yellow). From this abstract representation, which has smaller dimensions than the input, we reconstruct an output image of the same size as the input. The first and last layers use the Parametric Rectified Linear Unit activation function (PReLU [15]) to capture any negative value present in the intermediate flow field and to predict the final, real-valued (\mathbb{R}) variables in the fluid domain.

3.3 CFDNet Framework

CFDNet is a *physics solver–CNN model–physics solver* coupled framework instead of a pure surrogate. It is designed this way to incorporate domain-specific knowledge in the model inference and to reach the same convergence as the physics solver. Figure 4 depicts the CFDNet framework and below, we describe the three main stages of the framework and the two key ways we integrate domain-knowledge.

Warmup: The first flow field to start the CFD simulation is user-given, therefore it contains no *domain-specific* knowledge. So, we let the physics solver carry K initial iterations so that the fluid parameters adapt to the new flow case through Equations (1), (2), and (3). We determine K adaptively without user input by assessing the residual drop from the initial conditions which is an indicator of the evolution of the physics solver. A residual drop of one order of magnitude is sufficient for the fluid parameters close to the physical boundaries to capture the geometry and flow conditions of the new problem. We use this as our warmup stopping criteria.

CNN Inference: After warmup, we generate the input image, as described in Section 3.1. This input tensor which now has *domain-specific* knowledge of the new geometry and/or flow configuration is used as an input to the trained CNN model for inference. The model predicts the output tensor at steady state.

Iterative Refinement: The output of CNN inference may not satisfy known conservation laws. To ensure we meet the convergence constraints of the original algorithm, we feed the output of CNN inference as an input to the physics solver and perform M iterations till convergence. The expectation is that the overall

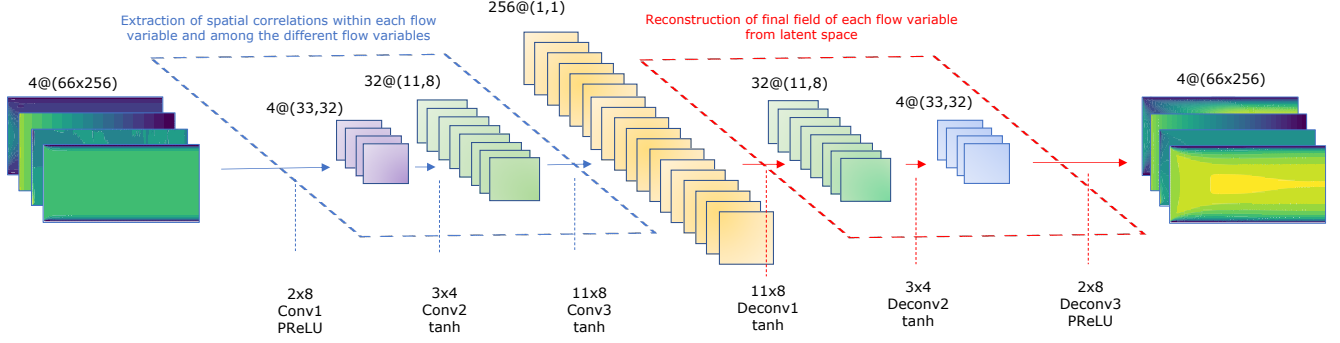


Figure 3: CNN architecture. The CNN is a symmetric 6-layer convolution-deconvolution neural network. The first three layers are convolution layers (Conv1 to Conv3) and they are followed by three transposed convolution layers (Deconv1 to Deconv3). The size of each filter is shown at the bottom of the figure along with the activation functions applied to each layer. A striding of the same size as the filter is applied in each layer.

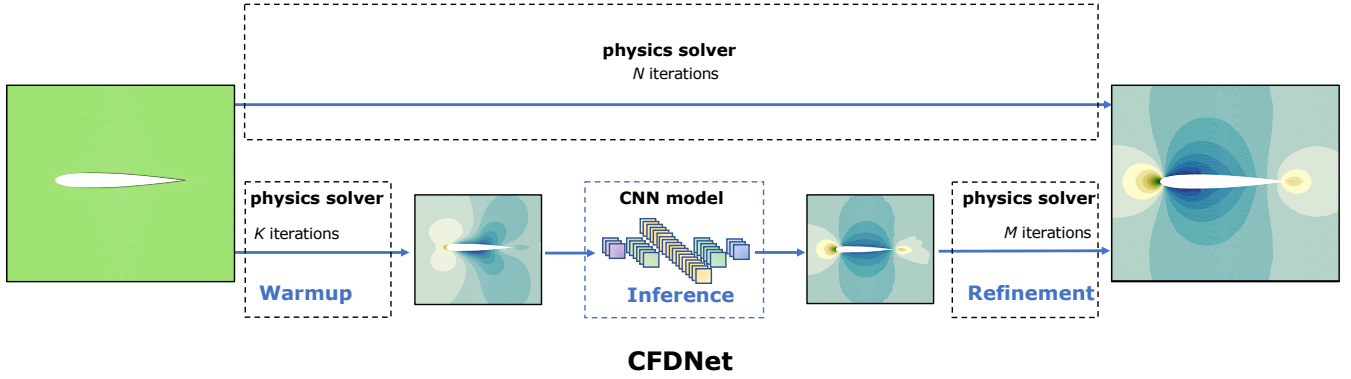


Figure 4: Comparison of the traditional physics solver simulation with CFDNet. CFDNet integrates the domain-specific physics solver for *warmup*, followed by the neural network for inferring the steady state, and the final iterative *refinement* stage to correct the solution of the CNN and satisfy the convergence constraints.

number of iterations to reach convergence will be less than the physics solver-only simulation (i.e., $K + M < N$) if the data-driven model is successful in *short-circuiting* the simulation.

3.4 Convergence Criteria and Error of CFDNet

If the CNN inference loss is less than the user-defined error tolerance compared to the ground truth data, CFDNet would return the CNN model's output tensor as the final steady-state flow field (i.e., the fluid flow parameters would have reached the convergence criterion defined by the user) bypassing the *refinement* stage. In this scenario, the CNN model is a *pure* surrogate of the RANS equations with the Spalart-Allmaras one-equation turbulence model. However, this approach has significant shortcomings.

First, the above-mentioned convergence criterion is based on error metrics found in the CFD literature [3]. We argue that evaluating the quality of the surrogate through these error metrics is sub-optimal for two main reasons – (i) they lack physical meaning, and (ii) metrics such as L_1 norm, mean absolute error (MAE),

and root mean squared error (RMSE) are ill-defined if no information about the order of magnitude of the quantity of interest is provided. Second, in many approaches, the conservation of mass and momentum is not a constraint in the optimization problem (this is true for our CNN model as well). Even though this optimization choice makes the model more generalizable (e.g., can be easily extended to support other simulations such as compressible flows), satisfying the conservation laws can be imperative for CFD practitioners/engineers. Third, ground truth data is typically not available since the goal is to predict the flow field of cases unseen during training. Therefore, evaluating the accuracy of the surrogate on error metrics with respect to the ground truth data lacks applicability and predictability in real scenarios.

Alternatively, other works [8, 36, 37] have considered finding a DNN-based substitute for only a single step of the iterative algorithm. For example, finding a surrogate of the Poisson solver in Eulerian flow simulations [8]. Because this approach modifies the original algorithm, one still needs to evaluate the quality of the result with the error metrics described above, therefore suffering from the same limitations.

We aim to circumvent the above limitations with the *refinement* stage in CFDNet and define its *dual convergence criteria* as follows.

- (1) In a traditional CFD simulation, convergence is typically achieved when the residual of all the variables has dropped 4-5 orders of magnitude [20] depending on the CFD practitioner. We adhere to the same constraints. Because the residual is referenced to the initial condition, and our initial condition in the *refinement* stage is a field close to the final solution, dropping the residual 4-5 orders of magnitude is sufficient to consider the cases fully converged. This is the first convergence criterion of CFDNet.
- (2) In addition to checking for residual convergence, it is a common practice among domain scientists/engineers to also ensure that the final solution satisfies conservation properties such as *conservation of mass*. We also adhere to this convergence constraint and this is the second convergence criterion of CFDNet.

When both *residual convergence* and *conservation laws* are satisfied, we claim that CFDNet satisfies the *convergence constraints* of the original physics solver and has a solution with 0% relative mean error (RME) with respect the physics solver solution. RME is defined as $\sum_{i=1}^{n_c} \frac{|\hat{y}_i - y_i|}{|y_i|}$ where n_c is the number of cells, \hat{y}_i is the single cell predicted value of the quantity of interest, and y_i is the single cell physics solver value. This metric is range and scale-invariant. Therefore, it is a better indicator of the quality of a prediction.

4 EXPERIMENT SETUP

In this section, we first describe the case studies including both laminar and turbulent flows for evaluating the potential of CFDNet as an accelerator. Then, we describe the construction of the training dataset for training the CNN model and then outline the training process.

4.1 Case Studies

We consider two types of fluid flows – wall-bounded flows and external aerodynamics which aim to stress different aspects of CFDNet. For the former, we keep the geometry constant while changing the flow configurations and for the latter, we keep the flow configuration constant while changing the geometry.

Wall-bounded Flows. The first case study is turbulent flow in a 2D channel. Many efforts have been made to understand wall-bounded turbulent flows, especially in confined geometries (e.g., pipe and channel designs). The study of these flow configurations is relevant for a broad range of Re^2 numbers, on which the final flow regime strongly depends. An application of CFDNet is to rapidly explore the design space for different ranges of Re .

In channel flow, transitional effects from laminar to turbulent flow occur around $Re \approx 3000$ [29]. Therefore, $Re = 4000$ is the lower bound in our design space to ensure fully developed turbulence flows. For $Re > 15000$, a different grid resolution than the chosen one should be used to have accurate CFD solutions. Hence, we scope the range $4000 < Re < 15000$ for our design exploration. We

train our CNN model on four different configurations: $Re = 4200$, 6800, 7500, and 12500, labeled as the training dataset A. For testing the interpolation and extrapolation performance of CFDNet, we choose $Re = 5600$ and $Re = 13750$ respectively.

Flow around solid bodies. Understanding flow around solid bodies has been an extensive field of research because of its relevance in aerodynamics design and industrial applications. It is common practice in aerodynamics design to explore the flow field around variations of the solid body to end up with one that satisfies the designer's performance requirements. For this reason, we apply CFDNet to flows around geometries that have not been seen during its training phase, in both laminar and turbulent flow conditions.

Training consists of flow around six different ellipses shown in Figure 5. The different ellipses are obtained by changing the aspect ratio (AR), that is, the ratio of the vertical to the horizontal semiaxis length from 0.1 to 0.7. For the turbulent case, Re is held constant at $6e5$ for all the experiments which yield a training dataset labeled B. Training dataset C has the same geometries as B but at laminar flow conditions ($Re = 30$).

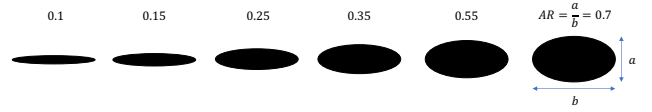


Figure 5: Ellipse geometries used for training in datasets B and C.

We test CFDNet on flows around an airfoil, cylinder, and an ellipse ($AR = 0.3$) not seen in the training phase as shown in Figure 6.

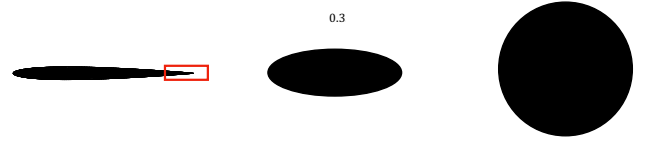


Figure 6: Airfoil, ellipse, and cylinder used for testing the CFDNet framework. We highlight the trailing edge of the airfoil as a new edge not seen by the network in training.

4.2 Physics Solver in OpenFOAM

The training dataset is generated by solving the RANS equations together with the Spalart-Allmaras one-equation model [33]. The grid is 64×256 , a proper grid dimensionality for all cases [2]. We use the incompressible solver `simpleFoam` from OpenFOAM v6 as the *physics solver* in this paper. The simulations are run till steady state reaching a residual value of $1e-6$ for the velocity and pressure and $1e-4$ for the modified eddy viscosity in the channel flow cases. For the flows around solid bodies, a residual value between $1e-5$ and $1e-6$ for the velocity and pressure and $1e-4$ for the modified eddy viscosity is considered acceptable. These tolerances are extended practice in CFD simulations [20].

Note that our goal in designing the CFDNet accelerator is to use it in tandem with other acceleration techniques commonly used in PDE simulations. Therefore, we use the GAMG solver for

²The Reynolds Number, or Re is a dimensionless quantity of significance in fluid mechanics and quantifies the flow conditions of the problem.

implicitly computing the pressure at every iteration, with a tolerance $1e-8$ and the `GaussSeidel` smoother from OpenFOAM. The velocity and the modified eddy viscosity are computed with the `smoothSolver` and the `GaussSeidel` smoother with the same tolerance as the pressure.

Architecture and Libraries. All the OpenFOAM simulations are run in parallel on a dual-socket Intel Xeon E5-2630 v3 processor (Haswell-EP). Each socket has 8 cores, for a total of 16 cores and a theoretical double-precision peak performance of 614.4 GFlop/s. We use the OpenMPI implementation of MPI integrated in OpenFOAM that is optimized for shared-memory communication. The grid domain is decomposed into 16 partitions using the OpenFOAM integrated Scotch partitioner and each partition is assigned to 1 MPI process that is pinned to a single core. The `numactl -localalloc` flag is set to bind each MPI process to its local memory.

4.3 Dataset Creation and Preprocessing

We create three distinct datasets based on the two case studies, as outlined in Section 4.1. Each dataset has a training set (used for training the CNN model), a validation set (to ensure the training is not over- or underfitting) and a test set, used for evaluating the performance of CFDNet. The recipe for creating the *training* set is as follows:

- (1) We perform the OpenFOAM simulation of all training flow configurations. During the simulation, we take snapshots of the flow field at every intermediate iteration, I_j (for flow configuration j) until the flow converges to steady-state. From each of these iteration snapshots of fluid parameters, we create the tensor image representation of our inputs to CFDNet as detailed in Section 3.1. The resulting number of images is N_j , where the first $N_j - 1$ are labeled as inputs and the steady-state image at iteration N_j is labeled as the output. Each of the first $N_j - 1$ iterations is independently mapped to steady-state N_j (i.e., every iteration of the simulation becomes a training sample whose output is the final steady solution). Therefore, the training set size for each dataset is $\sum_{j=1}^{n_f} N_j - n_f$, where n_f is the number of flow configurations considered in training (e.g., $n_f = 4$ for dataset A). Now, we add the initial conditions (iteration 0) also as a sample to the dataset and the training set size then becomes $\sum_{j=1}^{n_f} N_j$ for each dataset.
- (2) We *non-dimensionalize* all the samples. The fluid variables of all images are non-dimensionalized as follows: $\frac{\bar{u}}{\bar{U}_r}$, $\frac{\bar{v}}{\bar{U}_r}$, $\frac{\bar{p}}{\bar{U}_r^2}$ and $\frac{\bar{\nu}}{\bar{\nu}_r}$, where subscript r is a reference value. These reference values are flow configuration-specific and characterize each simulation.

The final training set sizes for datasets A, B and C are 6372, 14953, and 12988, respectively.

4.4 Training

We train three independent models, one for each dataset A, B, and C. The CNN (shown in Figure 3) is implemented using Keras [13] and training of the CNN is performed on a Tesla K40c NVIDIA GPU

using the TensorFlow 1.13 backend. No specific initialization is used in training. The batch sizes for training are chosen to be 1, 4, and 4 for the datasets A, B, and C respectively. The optimizer is RMSProp and the loss function is mean squared error (MSE). The learning rate is set to $7e-5$ with no decay for all training. For training set A, 15 epochs were sufficient to drop the training loss to $1.2e-6$ and validation loss to $3.2e-6$. For training set B, after 35 epochs the training loss reached $1.4e-4$ and the validation loss $2e-4$. For training set C, the training and validation losses dropped to $3.6e-5$ and $5.1e-5$, respectively after 29 epochs.

The reason for the higher loss in training sets B and C compared to A is because they contain a different set of geometries that present different flow regimes, even though the flow conditions are the same. Thicker ellipses ($AR = 0.55, 0.7$) present a significantly more complex, more non-linear flow regime. Since the geometry gradients are more accentuated, flow separation from the solid body occurs at these flow conditions. This causes a depression in the rear part of the solid body, leading to negative velocities (recirculation). In contrast, thinner ellipses ($AR = 0.1, 0.15$) present a smoother flow behavior. Therefore, the network not only needs to adapt to different grids, but it also needs to adapt to different physical phenomena. In training set A, the geometry and the flow physics are kept the same, therefore the network only has to adapt to a new flow configuration (a new Re number). The discrepancy between the losses of training set B and C can be explained through the eddy viscosity. Because of the different flow regimes among the ellipses, the turbulent intensity in the rear part of these solid bodies changes dramatically from ellipse to ellipse (higher in thicker ellipses). This is a physical phenomenon that does not occur in training set C since there is no turbulence.

5 RESULTS AND DISCUSSION

Once the CNN is trained and validated, to cover the entire spectrum of its predictive ability, we evaluate CFDNet on three use cases as outlined below.

Observed geometry, different flow conditions (OG-DF). We use CFDNet to accelerate the simulation of a flow on a geometry observed during training but on different flow configurations. Here, our flow case is channel flow and the CNN model is trained with dataset A. We test two different flow conditions – an input velocity to the channel ³ of 0.56 m s^{-1} and 1.375 m s^{-1} . With the former, we evaluate CFDNet’s capacity to interpolate to a new flow condition, whereas with the latter, we test its ability to extrapolate.

Subset geometry, same flow conditions (SG-SF). Test geometry is a *subset* of the training dataset. Here, we aim to test the ability of the CNN to interpret the edges of the new geometry which can be formed using a linear combination of the training geometries (i.e., interpolation). In this use-case, the flow conditions are kept the same throughout the experiment. Here, our flow case is external aerodynamics and the CNN models are trained with datasets B and C for turbulent and laminar flows respectively. The geometries used for training are depicted in Figure 5 and the test geometry is also an ellipse (with an unseen-in-training AR) shown in Figure 6.

³Note that referring to Re number or input velocity to the channel is interchangeable, since $Re = \frac{U_i L}{\nu}$, where U_i is the input velocity, L is the channel height (constant) and ν is the fluid laminar viscosity (constant).

Different geometry, same flow conditions (DG-SF). Test geometry is *different* from the training dataset. Here, we test CFDNet’s capacity for generalization where the edges of the test geometries have not been seen previously by the CNN. To underscore that, we reuse the CNN models trained for the previous use case SG-SF. We test the performance of CFDNet on the NACA0012 airfoil [32], the geometry of which follows the mathematical definition in [1] and a cylinder shown in Figure 6.

The accuracy and performance of CFDNet are evaluated by comparing it against the traditional physics solver-only simulation. We first evaluate the accuracy of our convolutional neural network prediction and the end-to-end CFDNet coupled framework. Then, we evaluate the performance of CFDNet and finally examine the importance of the *warmup* method in accelerating the convergence of CFDNet, especially for new test geometries.

5.1 Model and CFDNet Accuracy

One of the goals of this paper is to accelerate flow simulations using deep learning without relaxing the convergence constraints of the physics solver. Since the network is trained to predict the final steady-state (iteration N), we hypothesize that the inference is “close” to the final solution. In this section, we test this hypothesis by comparing the CNN model’s output with CFDNet’s output, i.e., compare the solution with and without *refinement*. At the same time, we also compare both to the physics solver solution qualitatively and quantitatively.

Figure 7 shows the qualitative results for the OG-DF interpolated case. The leftmost column (a) presents the velocity field after *warmup + inference* (i.e., the CNN model’s output). We can observe the similarity between the output from the CNN and the physics solver solution in the third column (c). However, this is not the case for the DG-SF airfoil at $Re = 6e5$. We can see that the CNN predicted pressure field in Figure 8 (a) qualitatively differs from the physics solver solution (c) mostly in the front part of the airfoil. Similar to the airfoil, we can visually conclude that the prediction of the modified eddy viscosity for the DG-SF cylinder in Figure 9 with the CNN (a) is qualitatively far from the physics solver (c). This discrepancy in the quality of the prediction between the OG-DF channel flow case and the DG-SF airfoil/cylinder case is expected from the training results (refer to Section 4.4) and the challenge of each test case individually - interpolate a new flow condition vs. extrapolate to an unseen geometry.

Quantitatively, the per-cell absolute error between the CNN prediction and the physics solver solution can be seen in Figure 7 (d) for OG-DF channel flow case. The per-cell absolute error is 2 to 3 orders of magnitude lower than the variable value, yielding a RME less than 2% for all flow variables, which is in line with acceptable RME reported in the literature, as seen in Table 2. This suggest that the CNN (i.e., *warmup + inference*) is a promising approach as a pure surrogate when the geometry is the same (OG-DF) but not otherwise.

Recall from Section 3.4 that the quantitative analysis through the above error metrics alone is insufficient. Moreover, Table 2 shows that there is no consensus among SOTA approaches on the best error metric to adopt. Therefore, we conduct the conservation of mass *check*. Because the flow is incompressible, the velocity field

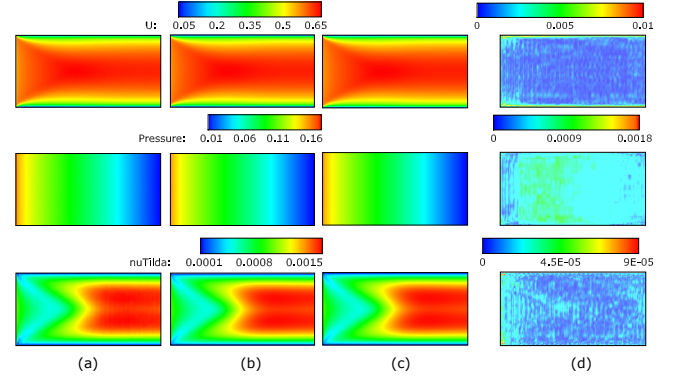


Figure 7: Velocity field in $m s^{-1}$ (top), Kinematic pressure field in m^2/s^2 (middle), and Modified eddy viscosity field in m^2/s (bottom) for channel flow at $Re = 5600$. (a) *warmup + inference* (no refinement), (b) CFDNet, (c) physics solver in OpenFOAM, and (d) per-cell absolute error between (a) and (c).

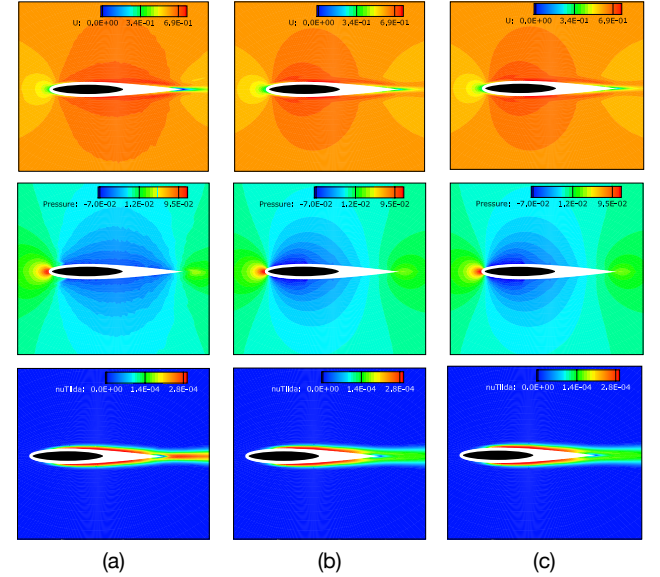


Figure 8: Velocity field in $m s^{-1}$ (top), Kinematic pressure field in m^2/s^2 (middle), and Modified eddy viscosity field in m^2/s (bottom) around the airfoil at $Re = 6e5$. (a) *warmup + inference* (no refinement), (b) CFDNet, and (c) physics solver in OpenFOAM.

has to satisfy the conservation of mass Equation (1), i.e., $\nabla \cdot U = 0$. OpenFOAM provides a tool to numerically compute the divergence of the velocity for a user-given flow field. Table 3 compares the result of the tool for the flow fields from CNN model-only (*warmup + inference*), CFDNet (*warmup + inference + refinement*), and the physics solver-only simulations.

Because analytically the divergence of the velocity needs to be exactly 0, numerically it should be close to the machine round-off errors. Table 3 illustrates how CFDNet and the physics solver yield a divergence-free field (i.e., both satisfy conservation of mass), whereas the CNN model’s output is far from the expected tolerance.

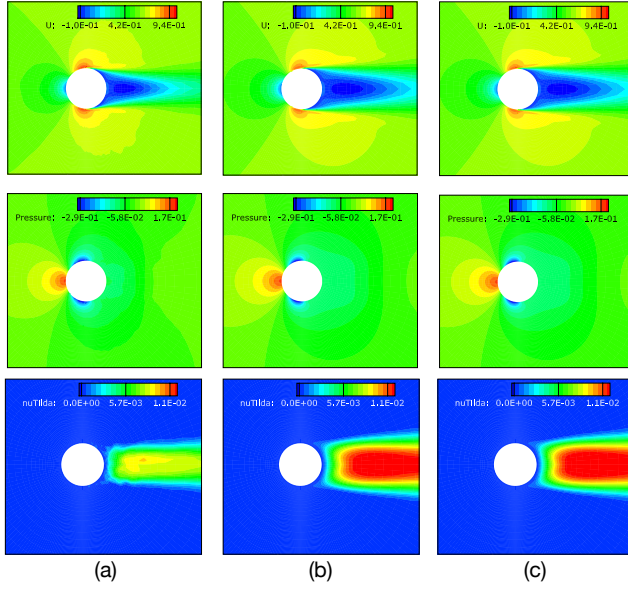


Figure 9: Velocity field in m s^{-1} (top), Kinematic pressure field in m^2/s^2 (middle), and Modified eddy viscosity field in m^2/s (bottom) around the cylinder at $Re = 6e5$. (a) *warmup* + *inference* (no *refinement*), (b) CFDNet, and (c) physics solver in OpenFOAM.

Table 2: Errors reported in the literature. Different works have considered different Quantities of Interest (QoI). There is no consensus for an acceptable magnitude of error, and the errors reported as acceptable vary by work.

Network	Metric	Best	QoI
Guo et al. [14]	RME	1.76%	\bar{U}
Thuerey et al. [35]	RME	<3%	\bar{U}, \bar{p}
Smart-fluidnet [8]	MAE	9e-3	ρ
TBNN [20]	RMSE	0.08	Reynolds Stresses
Tompson et al. [36]	L_2 norm	0.872	$\nabla \cdot U$
Maulik et al. [22]	L_1 norm	<2e-4	v_t
CFDNet (this paper)	RME	0%	\bar{U}, \bar{p} and \bar{v}

Table 3: Mass local error. The value reported is the average over all the cells. The CNN model does not result in a divergence-free field. This is a major motivation for refinement after the initial warmup and inference steps. After the final refinement, the field is divergence-free.

Test Case	warmup+inference only	CFDNet	physics solver
channel flow $Re = 5600$	3.12e-4	<1e-10	<1e-10
airfoil $Re = 6e5$	8.15e-3	<1e-10	<1e-10
cylinder $Re = 6e5$	9.8e-3	<1e-10	<1e-10

Even with a CNN model’s prediction - for the channel flow case - yielding a 2% RME, the *iterative refinement* becomes necessary to satisfy the conservation laws and meet the convergence constraints of the original physics solver.

5.2 Performance Analysis

The time to solution of CFDNet is the sum of the *warmup*, *inference*, and *refinement* times which are defined as follows.

Warmup time, t_{warmup} is the time for the physics solver to drop the error from the initial condition one order of magnitude (i.e., K iterations). The physics solver is run in parallel as described in Section 4.2.

Inference time, t_{infer} is the sum of the times spent on each of the following steps: (1) `MPI_Gather` on the master process to get all the flow variables values from each MPI process after warmup, (2) construction of the input tensor image, (3) CNN inference of the output (the inference is computed on the CPU whose specifications are detailed in Section 4.2), (4) `MPI_Scatter` to distribute the output tensor image back to the MPI processes.

Refinement time, t_{refine} is the time for the physics solver to further drop the error from the new initial conditions (i.e., the output tensor after inference) four orders of magnitude for all variables in test sets B and C. For test set A, we drop 5 orders of magnitude for the velocity and pressure fields and 4 orders of magnitude for the eddy viscosity. Note that different convergence criteria are used for each test set to correspond to the respective residuals (tolerances) used to create the training sets. *Refinement*, as the *warmup*, is run in parallel.

Figure 10 compares the CFDNet time (broken down by t_{warmup} , t_{infer} , and t_{refine}) to the time the traditional physics solver implementation takes to drop the residual to the same user-defined tolerance defined above for each dataset. The values inside the bars are the number of iterations required by the physics solver in each stage (i.e., K for warmup, M for inference, and N for the physics-only solver). We first focus on the end-to-end CFDNet performance and then, evaluate specifically, the importance of warmup in embedding domain-knowledge.

CFDNet vs physics solver. The leftmost stacked bar (a) for each test case represents the time taken by CFDNet and the rightmost bar (c) is the time taken by the OpenFOAM physics solver to meet the convergence criteria. Across the board, CFDNet accelerates the simulations by a factor of $1.9 - 7.4\times$. The highest performance gain is achieved for the OG-DF test cases followed by SG-SF and DG-SF last. Compared to the physics solver-only simulation which requires 1578 iterations (1588 for extrapolated flow conditions) to drop the residual to $1e-5$ for OG-DF, CFDNet only takes one CNN inference and 319 iterations (426 for extrapolated case) to reach the same convergence constraints. SG-SF and DG-SF evaluate the framework’s ability to generalize on unseen geometries (where the former uses a subset geometry while the latter has edges not included in the training datasets). We observe that CFDNet outperforms the traditional physics solver on both use cases requiring fewer total number of iterations to drop the residual to meet the convergence constraints.

Recall from earlier that in contrast to the OG-DF channel flow case, the standalone network has difficulty inferring the right flow field around the airfoil and cylinder in the DG-SF flow cases. This is not surprising since prediction on a different geometry (whether it is inter- or extrapolated) is much more challenging than when the geometry is kept fixed. However, it is interesting to note that the network finds the cylinder geometry particularly challenging

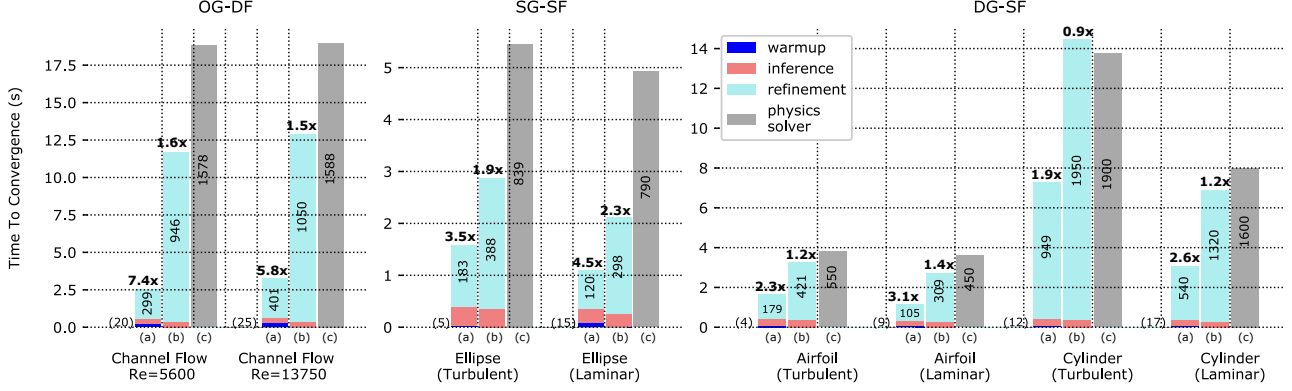


Figure 10: Breakdown of running time to convergence for each test case. Time to convergence of (a) CFDNet w/ *warmup*, (b) CFDNet w/o *warmup*, and (c) the physics solver. The values inside the bars are the number of iterations it took for that stage. The number of *warmup* iterations is indicated in parenthesis. At the top of each column is the speedup with respect to the physics solver.

to accelerate compared to the flow around the airfoil and interpolated ellipse. Even though a cylinder is simply a special case of an ellipse from a geometrical perspective, the physics in the rear part of the cylinder is more non-linear (a large recirculation area) and therefore, more refinement iterations than the other cases are required. Nevertheless, CFDNet still outperforms the physics solver by 1.9 – 2.6× for the DG-SF cylinder case. These results show CFDNet’s potential to generalize to geometries whose fluid flows are significantly different from those in the training set while outperforming state-of-the-art physics solvers.

Importance of *warmup*. We now compare the results presented in the previous section when the inference is done without *domain-specific* knowledge (no *warmup*). The (b) bars in Figure 10 show the time taken by CFDNet when the *warmup* method is not applied. Without *warmup*, the speedups drop significantly to 1.2 – 2.3×.

In CFD simulations, the *domain-specific* knowledge is not embedded until the numerical scheme interacts with the values in the boundary conditions. For example, the user-given initial condition is domain-blind. This is the objective of *warmup*: let the solver carry (K) initial iterations so that *domain-specific* knowledge (e.g., geometry definition, flow condition, etc) is embedded in the input tensor.

In the flow around solid bodies case, having the geometry entirely defined in the input tensor is critical for the network to predict the right flow regime. This is specially true for the DG-SF turbulent cylinder. Without *warmup*, the network predicts a field quantitatively⁴ poor compared to the physics solver solution. The *iterative refinement* uses a lot of iterations to meet the physics solver convergence constraints. For this reason, we even observe a slowdown compared to the physics solver without *warmup*. To summarize, across all the flow configurations tested in this paper, *warmup* only takes 1 – 2% of the overall iterations of the physics solver while yielding significant speedups of 1.9 – 4.6×.

⁴Our computed RME in this case are 110% for the velocity, 251% for the pressure, and 80% for the modified eddy viscosity.

6 RELATED WORK

Computational Fluid Dynamics. Direct Numerical Simulation (DNS) is an attempt to solve the discretized Navier-Stokes equations accounting for all time and length scales of turbulence [29]. This leads to very fine meshes and small timesteps which make DNS computationally intractable for several flows. Reynolds Averaged Navier-Stokes (RANS) equations [30] time-average the effects of turbulence at the expense of yielding a non-closed equation. Closing the equation has been extensively done through modeling [33]. However, even with RANS coupled with turbulence modeling, exploring different design alternatives for complex flows and geometries is computationally expensive. To circumvent this, it is common to use surrogates - imperfect but computationally inexpensive models to accelerate design space exploration for industrial applications.

Accelerating Fluid Simulations with Surrogates. Shape design optimization relies on the minimization of an objective function. Gaussian Processes (GP) approximate this objective function to be computationally tractable. However, these models rely on a specific geometry parametrization [16] which limits its generalizability. Other regression techniques, such as MARS and polynomial regressions [4, 10, 12] only predict the velocity or pressure fields on particular points on the surface of the solid body, in order to reduce the dimensional complexity of the problem. In this paper, we take an alternate approach with neural networks that predicts the velocity and the pressure fields for all points in the flow field.

Accelerating Fluid Simulations with Neural Networks. In the past few years, several researchers have leveraged neural networks to accelerate fluid dynamics simulations. Tompson et al. accelerate Eulerian fluid simulations by replacing the Poisson solver step in an Eulerian flow iterative solver instead of finding an end-to-end mapping to calculate the divergence-free velocity field [36]. Alternatively, Guo et al. find a real-time solution to viscous laminar flows around solid objects [14]. However, the above approaches have elemental constraints. First, Eulerian fluid simulations ignore second-order velocity derivatives in the Navier-Stokes equations.

Second, viscous laminar flow approaches are ambiguous if they are to be expanded to turbulent flows which are intrinsically chaotic and much harder to resolve [29].

There have been recent attempts to find neural network-based accelerators for turbulent flows. The results are promising but the neural networks predict only a subset of the flow variables. Maulik et al. predict the eddy viscosity field and not other flow properties such as velocity and pressure fields [22]. Thuerey et al. use a novel input-output representation but their approach does not account for the eddy viscosity field [35]. More importantly, the network prediction is limited to the fluid domain closest to the solid body, so it remains an unknown how the network would perform in the freestream, where the boundary conditions are set which define the flow configuration. Another limitation of [35] is that the network is used as a final, end-to-end surrogate. Even though the results reported are promising (RME less than 3%) and the surrogate approach provides real-time solutions, the geometries in the training and prediction stages are the same (airfoils). It remains unclear how the surrogate would perform on different geometries. CFDNet injects the neural network-based mapping back into the physical solver to enable both extrapolation and generalization with the same model without relaxing the convergence constraints.

7 CONCLUSIONS

This paper has shown that coupling of a RANS fluid flow simulation with a convolutional neural network, an approach called CFDNet, can significantly accelerate the convergence of the overall scheme. CFDNet speeds the simulations up by a factor of $1.9 - 7.4\times$ on both steady laminar and turbulent flows on a variety of geometries, without relaxing the convergence constraints of the physics solver. To evaluate the model's capacity for generalization and extrapolation, it is tested across a range of scenarios and geometries, including channel flow, ellipses, airfoils, and cylinders. In general, the model performs well and demonstrates a capacity to make accurate predictions even for geometries unseen during training.

CFDNet indicates that coupling physical models with data-driven machine learning models is a promising approach for accelerating the convergence of simulations. However, more work remains for this method to be a widely used tool for predictive engineering. The explorations shown in this paper are focused on fluid-flow simulations, in particular, RANS models, but the fundamental approach is not expected to be limited to a particular model or physical domain. For example, it is expected that CFDNet could be successfully applied, with minimal effort, to Large Eddy Simulations (LES). In principal, any discretized field of inputs should be amenable to the approach outlined in this paper, and so a wide variety of domains in scientific computing, such as molecular dynamics, material science, etc., could be considered. This approach may not be limited to physical simulations, and may be useful as a preconditioner or as a replacement for algebraic multigrid or other numerical linear algebra solvers. Regardless, the generalizability of the network architecture remains an important open question. Many CNN networks have demonstrated efficacy across a variety of related tasks with only minimal re-training necessary. The capability of transfer learning or at least leveraging a similar network architecture for a wide range of tasks is an important future exploration.

Finally, while the results in this paper maintained the accuracy of the underlying simulation, a user of CFDNet will not desire to run a traditional simulation in parallel to check the simulation results. Thus, more research should be performed on quantifying the uncertainty and robustness of coupling DL and traditional physical simulations.

8 ACKNOWLEDGMENT

This work was partly supported by the National Science Foundation (NSF) under the award number 1750549. Any opinions, findings and conclusions expressed in this material are those of the authors and do not necessarily reflect those of NSF. We thank Dr. Ferran Martí, Senior Aerodynamicist at Boeing, for his help in sectioning and defining iterative time-marching numerical schemes and in the design of the airfoil's computational grid. ©2020 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

REFERENCES

- [1] [n.d.]. NACA0012. https://turbmodels.larc.nasa.gov/naca0012_val.html.
- [2] [n.d.]. NACA0012 grids. https://turbmodels.larc.nasa.gov/naca0012_grids.html.
- [3] MYM Ahmed and N Qin. 2009. Surrogate-based aerodynamic design optimization: use of surrogates in aerodynamic design optimization. In *International Conference on Aerospace Sciences and Aviation Technology*, Vol. 13. The Military Technical College, 1–26.
- [4] Vladimir O Balabanov, Antony A Giunta, Oleg Golovidov, Bernard Grossman, William H Mason, Layne T Watson, and Raphael T Haftka. 1999. Reasonable design space approach to response surface approximation. *Journal of Aircraft* 36, 1 (1999), 308–315.
- [5] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. 2014. Searching for Exotic Particles in High-Energy Physics with Deep Learning. *Nature Commun.* 5 (2014), 4308. <https://doi.org/10.1038/ncomms5308> arXiv:hep-ph/1402.4735
- [6] Cx K Batchelor and GK Batchelor. 2000. *An introduction to fluid dynamics*. Cambridge university press.
- [7] Joseph Boussinesq. 1877. *Essai sur la théorie des eaux courantes*. Impr. nationale.
- [8] Wenqian Dong, Jie Liu, Zhen Xie, and Dong Li. 2019. Adaptive neural network-based approximation to accelerate eulerian fluid simulation. (2019), 7.
- [9] N Benjamin Erichson, Michael Muehlebach, and Michael W Mahoney. 2019. Physics-informed autoencoders for lyapunov-stable fluid flow prediction. *arXiv preprint arXiv:1905.10866* (2019).
- [10] Hongbing Fang, Masoud Rais-Rohani, Zheny Liu, and MF Horstemeyer. 2005. A comparative study of metamodeling methods for multiobjective crashworthiness optimization. *Computers & Structures* 83, 25–26 (2005), 2121–2136.
- [11] UKNG Ghia, Kirti N Ghia, and CT Shin. 1982. High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *Journal of computational physics* 48, 3 (1982), 387–411.
- [12] Anthony A Giunta. 1997. *Aircraft multidisciplinary design optimization using design of experiments theory and response surface modeling methods*. Ph.D. Dissertation. Virginia Tech.
- [13] Antonio Gulli and Sujit Pal. 2017. *Deep learning with Keras*. Packt Publishing Ltd.
- [14] Xiaoxiao Guo, Wei Li, and Francesco Iorio. 2016. Convolutional neural networks for steady flow approximation. (2016), 481–490.
- [15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. (2015), 1026–1034.
- [16] James Hensman, Nicolo Fusi, and Neil D Lawrence. 2013. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835* (2013).
- [17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems* 25, F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105. <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
- [18] Myoungkyu Lee, Nicholas Malaya, and Robert D Moser. 2013. Petascale direct numerical simulation of turbulent channel flow on up to 786K cores. (2013), 1–11.

- [19] Zeljko Lilek, Samir Muzaferija, and Milovan Perić. 1997. Efficiency and accuracy aspects of a full-multigrid simple algorithm for three-dimensional flows. *Numerical Heat Transfer* 31, 1 (1997), 23–42.
- [20] Julia Ling, Andrew Kurzawski, and Jeremy Templeton. 2016. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics* 807 (2016), 155–166.
- [21] Yunjie Liu, Evan Racah, Prabhat, Joaquin Correa, Amir Khosrowshahi, David Lavers, Kenneth Kunkel, Michael Wehner, and William Collins. 2016. Application of Deep Convolutional Neural Networks for Detecting Extreme Weather in Climate Datasets. *arXiv preprint arXiv:1605.01156* (2016).
- [22] Romit Maulik, Himanshu Sharma, Saumil Patel, Bethany Lusch, and Elise Jennings. 2019. Accelerating RANS turbulence modeling using potential flow and machine learning. *arXiv preprint arXiv:1910.10878* (2019).
- [23] Bahareh Mostafazadeh, Ferran Marti, Feng Liu, and Aparna Chandramowlishwaran. 2018. Roofline Guided Design and Analysis of a Multi-stencil CFD Solver for Multicore Performance. In *Proc. 32nd IEEE Int'l. Parallel and Distributed Processing Symp. (IPDPS)*. Vancouver, British Columbia, Canada, 753–762.
- [24] Bahareh Mostafazadeh Davani, Ferran Marti, Behnam Pourghassemi, Feng Liu, and Aparna Chandramowlishwaran. 2017. Unsteady Navier-Stokes Computations on GPU Architectures. In *23rd AIAA Computational Fluid Dynamics Conference*. 4508.
- [25] Mustafa Mustafa, Deborah Bard, Wahid Bhimji, Zarija Lukić, Rami Al-Rfou, and Jan M Kratochvil. 2019. CosmoGAN: creating high-fidelity weak lensing convergence maps using Generative Adversarial Networks. *Computational Astrophysics and Cosmology* 6, 1 (2019), 1–13.
- [26] Samir Muzaferija and David Gosman. 1997. Finite-volume CFD procedure and adaptive error control strategy for grids of arbitrary topology. *Journal of computational physics* 138, 2 (1997), 766–787.
- [27] Atsushi Okajima, Donglai Yi, Atsushi Sakuda, and Tomohito Nakano. 1997. Numerical study of blockage effects on aerodynamic characteristics of an oscillating rectangular cylinder. *Journal of wind engineering and industrial aerodynamics* 67 (1997), 91–102.
- [28] Suhas Patankar. 2018. *Numerical heat transfer and fluid flow*. CRC press.
- [29] Stephen B Pope. 2001. Turbulent flows.
- [30] Osborne Reynolds. 1895. Iv. on the dynamical theory of incompressible viscous fluids and the determination of the criterion. *Philosophical transactions of the royal society of london.(a.)* 186 (1895), 123–164.
- [31] Francesco Salvatore, Matteo Bernardini, and Michela Botti. 2013. GPU accelerated flow solver for direct numerical simulation of turbulent flows. *J. Comput. Phys.* 235 (2013), 129–142.
- [32] Hua Shan, Li Jiang, and Chaoqun Liu. 2005. Direct numerical simulation of flow separation around a NACA 0012 airfoil. *Computers & fluids* 34, 9 (2005), 1096–1114.
- [33] Philippe Spalart and Steven Allmaras. 1992. A one-equation turbulence model for aerodynamic flows. In *30th aerospace sciences meeting and exhibit*. 439.
- [34] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going Deeper with Convolutions. In *CVPR 2015*. <http://arxiv.org/abs/1409.4842>
- [35] Nils Thuerey, Konstantin Weißenow, Lukas Prantl, and Xiangyu Hu. 2019. Deep Learning Methods for Reynolds-Averaged Navier–Stokes Simulations of Airfoil Flows. *AIAA Journal* (2019), 1–12.
- [36] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. 2017. Accelerating eulerian fluid simulation with convolutional networks. (2017), 3424–3433.
- [37] Linyang Zhu, Weiwei Zhang, Jiaqing Kou, and Yilang Liu. 2019. Machine learning methods for turbulence modeling in subsonic flows around airfoils. *Physics of Fluids* 31, 1 (2019), 015105.