UNIVERSITY OF CALIFORNIA,
IRVINE

Deep Learning Algorithms for Accelerating Fluid Simulations

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Mechanical and Aerospace Engineering

by

Octavi Obiols-Sales

Dissertation Committee:
Associate Professor Aparna Chandramowlishwaran, Chair
Assistant Professor Ramin Bostanabad
Associate Professor Sameer Singh

2022

DEDICATION

To my family

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

First and foremost, I thank Professor Aparna Chandramowlishwaran for taking me as her PhD student. I am extremely grateful for her trust. I switched advisors after my Master's degree in Mechanical and Aerospace Engineering and she accepted me even though I had no background in computer science or artificial itelligence. She trusted me and gave me the possibility to explore together the fascinating world of artifical intelligence for science. This dissertation, together with the three scientific publications that come with it, are the result of this magnific collaboration. Not everybody in her position would have taken this risk, and so I appreciate it immensely. Thank you for everything.

Second, I want to thank the National Science Foundation (NSF) because they supported this work under the award number 1750549. This dissertation and research would not exist without this funding and for it I am incredibly grateful.

I thank Dr. Nicholas Malaya and Dr. Abhinav Vishnu for being two excellent co-advisors. Since my internship at AMD, we have kept an intense collaboration. Their guidance and technical advices have been key for my intellectual development and research successes that are also theirs. I will be forever grateful.

I thank Professor Roger Rangel for bringing me to UCI. He trusted me and accepted my application for the Balsells Fellowship, which allowed me to conduct my Masters and PhD studies in an excellent environment. I thank the Balsells Fellowship program (Pete Balsells, UCI, and the Generalitat de Catalunya) for giving me the chance to study at UCI. Without them, my time at UCI and this dissertation would not exist. I will be forever grateful.

I want to thank my labmates at UC Irvine. I thank Ferran, Laleh, Rohit, Behnam, Shu-Mei, and Hengjie. I cherish all the happy memories with the group.

I thank my family. I thank my parents, Lluis and Carme, and my sister, Alba, for being an unconditional support. Thank you for your endless love and support that has given me the strength to continue during the dark times. I could have not asked for a better family.

And last but definitely not least, thank you, Maria, the love of my life. This dissertation and everything that comes next are definitely also yours. You are the best partner anyone could ask for and I am incredibly grateful you chose to share your life with me. I love you.

VITA

Octavi Obiols-Sales

EDUCATION

Doctor of Philosophy in Mechanical and Aerospace Engineering **2022**
University of California, Irvine *Irvine, CA*

Master of Science in Mechanical and Aerospace Engineering **2017**
University of California, Irvine *Irvine, CA*

Master of Science in Aeronautical Engineering **2015**
Universitat Politecnica de Catalunya - BarcelonaTech *Barcelona, Spain*

Bachelor of Science in Aerospace Engineering **2014**
Universitat Politecnica de Catalunya - BarcelonaTech *Barcelona, Spain*

RESEARCH EXPERIENCE

Graduate Research Assistant **2015–2022**
University of California, Irvine *Irvine, CA*

REFEREED CONFERENCE PUBLICATIONS

CFDNet: a Deep Learning-Based Accelerator for Fluid Simulations **January 2020**
International Conference on Supercomputing (ICS)

SURFNet: Super-Resolution of Turbulent Flows with Transfer Learning Using Small Datasets **September 2021**
Parallel Architectures and Compilers Techniques (PACT)

ADARNet: Deep Learning Predicts Adaptive Mesh Refinement **January 2022**
arXiv

SOFTWARE

CFDNet, SURFNet, and ADARNet `https://github.com/oobiols/staidy.git`
Source code for CFDNet, SURFNet, and ADARNet

ABSTRACT OF THE DISSERTATION

Deep Learning Algorithms for Accelerating Fluid Simulations

By

Octavi Obiols-Sales

Doctor of Philosophy in Mechanical and Aerospace Engineering

University of California, Irvine, 2022

Associate Professor Aparna Chandramowlishwaran, Chair

Computational fluid dynamics (CFD) is the de-facto method for solving the Navier-Stokes equations, the set of partial differential equations that describe most laminar and turbulent flow problems. Solving this system of equations requires extensive computational resources; hence significant progress for scaling CFD simulations has been made with advancements in high-performance computing. However, the CFD community has mainly focused on developing high-order accurate methods instead of designing algorithms that harness the full potential of the new hardware. Moreover, current CFD solvers do not effectively utilize heterogeneous systems, where graphics processing units (GPUs) accelerate multi-core central processing units. At the same time, deep learning (DL) algorithms, whose training and inference stages map well to GPUs, have revolutionized the fields of computer vision and natural language processing. In this dissertation, we explore and propose novel algorithms to improve the performance and productivity of CFD solvers using DL.

First, we present CFDNet, a new convolutional neural network-based framework that accelerates laminar and turbulent flow simulations. Early works on DL+CFD approaches proposed surrogates that predict the flow field without any guarantees of satisfying the physical laws. Instead, we design CFDNet as an *accelerator* that reaches the same convergence guarantees as traditional first principles-based methods with fewer iterations. As a result, CFDNet

achieves $1.9 - 7.4\times$ speedups without compromising the quality of the solution of the physical solver in both laminar and turbulent flow problems for different configurations (such as channel flow and flow around an airfoil). CFDNet is the first DL-based accelerator for fluid simulations and presents three advantages: (a) it can be used in tandem with other acceleration techniques, such as multigrid solvers and parallelization, (b) it is amenable to any time-marching scheme, and (c) it is a DL module that can be plugged into any existing physical solver.

Like classical DL algorithms, CFDNet relies on training on large-scale datasets. Hence, it becomes impractical for high-resolution problems due to computationally prohibitive data collection and training. To overcome this limitation, we employ the idea of transfer learning (that is, reusing a model trained with a large number of samples for a task where data is scarce) and propose SURFNet: a transfer learning-based framework to accelerate high-resolution simulations. SURFNet performs data collection and training mostly at low resolution (64×256) while being evaluated at high resolutions (up to 2048×2048), improving the scalability of DL algorithms for CFD. SURFNet achieves a constant $2\times$ acceleration across different unseen-during-training flow configurations (such as symmetric and non-symmetric airfoils), and resolutions, showcasing resolution-invariance up to 2048×2048 spatial resolutions - significantly larger than those attempted in the literature.

SURFNet accelerates fluid simulations based on uniform meshes. However, since different regions of the domain present different flow complexity, we do not require uniform numerical accuracy throughout the domain. Adaptive mesh refinement (AMR) is an iterative technique that refines the mesh only in those regions that require higher numerical accuracy, and CFD solvers use it extensively for scalability. We propose ADARNet, a DL algorithm that predicts a non-uniform output and decides the final resolution of different domain regions in a single shot. Hence, ADARNet marries the advantages of DL (one-shot prediction) and AMR solvers (non-uniform refinement) to present a novel algorithm that outperforms both. Due

to ADARNet's ability to super-resolve only regions of interest, it predicts the same target 1024×1024 spatial resolution $7 - 28.5\times$ faster than state-of-the-art DL methods (which perform uniform super-resolution) and reduces the memory usage by $4.4 - 7.7\times$, showcasing improved scalability.

CFDNet, SURFNet, and ADARNet are hybrid DL-CFD frameworks that collectively improve the state-of-the-art. First, CFDNet is a DL-based accelerator for iterative numerical schemes. Second, SURFNet scales CFDNet to high resolutions and allows acceleration of real-world aerospace design scenarios. Third, ADARNet is a direct method for AMR that offers high-resolution accuracy with significantly less compute and memory resources. The code for these frameworks is open-source and can be found in: https://github.com/oobiols/staidy.git.

Chapter 1

Introduction

The Navier-Stokes (NS) equations, a set of second-order, nonlinear partial differential equations (PDEs) derived from Newton's second law of motion, describe most laminar and turbulent flows and lie at the core of fluid mechanics research. However, these equations have not yet been proved to have an analytical solution under all potential situations (for instance, turbulent 3-D flows). Therefore, the fluid mechanics community has extensively solved these problems via finite differences, mainly via the finite volume method (FVM). FVM discretizes the NS equations both in time and space and makes it possible to find a finite (numerical and approximate) solution to problems that, a priori, do not have an analytical (exact) solution.

One could try to find this numerical solution via pen and paper, but this approach becomes impractical due to the time complexity of these problems. Instead, we find this numerical solution via physical solvers: practitioner-generated codes that solve the NS equations numerically. Hence, computational fluid dynamics (CFD) is the set of all numerical schemes and iterative solvers that solve the discretized version of the NS equations. Nowadays, CFD is the primary source of fluid mechanics research, results, and analysis. In the last 30 years, the community has put considerable effort into improving the scalability and performance

of these solvers.

Direct numerical simulation (DNS) attempts to solve the discretized NS equations accounting for all turbulence time and length scales [1]. Unfortunately, this leads to very fine meshes and small timesteps, making DNS computationally intractable for several flows. To ease the tractability of these problems, Reynolds averaged Navier-Stokes (RANS) equations [2] time-average the effects of turbulence and often model those effects that are out of the scope of the average behavior. We describe the RANS equations and subsequent models in Chapter 2 of this thesis.

However, RANS is still computationally too expensive for extensive design space exploration if approached as-is. Solving the NS equations at every grid cell[1] is a daunting task. Therefore, the latest advances in current acceleration techniques, such as parallelization and multigrid solvers, have allowed the resolution of problems that seemed impossible several years ago [3]. However, it would be completely wrong to think that CFD represents a mature technology now. There are still many open questions, such as efficient solution techniques for viscous flows, design optimization/exploration, and the coupling between CFD and other disciplines [4, 5]. Because of the recent success of deep learning (DL) algorithms in computer vision (CV) and natural language processing (NLP)[6, 7], and because of the increasing heterogeneity of modern high-performance computing (HPC) systems, we hypothesize that these algorithms can play a key role in the future development of CFD simulations. In this thesis, we present new acceleration methods that leverage DL algorithms as a revolutionary and novel tool to use together with current acceleration techniques such as multigrid and parallelization [8–11].

[1] Typical grid sizes for accurate RANS solutions can range 1-20M.

1.1 The performance challenge of CFD

The ability to simulate turbulent flows using CFD has progressed rapidly over the last several decades and has fundamentally changed the aerospace design process [4, 12, 13]. The evolution of physics-based technologies has been crucial in providing advanced simulation capabilities. However, the last decade has seen stagnation in these capabilities, summarized by two main factors. First, the most significant challenge faced by the CFD community is, despite the improvement in the last several years, the inability of current methods to accurately predict turbulent-separated flows [14]. Second, HPC hardware is progressing rapidly and leading the community to rethink CFD algorithms and software. Most of the currently used algorithms and numerical schemes were developed several years ago and were not designed to exploit an increasing number of heterogeneous proessors [15]. The need for new algorithms - which is the focus of this research - is critical to harness the potential of massively parallel, heterogeneous HPC architectures with accelerators such as graphics processing units (GPUs) and field-programmable gate arrays [16] to explain physical systems of interest that remain currently unexplored.

While the vast majority of CFD codes (for instance, NASA's CFL3D [17], OVERFLOW [18], and FUN3D [19], or others such as OpenFOAM [20]) only run on central processing units (CPUs), DL algorithms on GPUs constantly outperform CPUs [21] at both training and inference. Due to its fast-paced growth, technologies that will prevail in HPC are challenging to predict. However, it is expected to require new algorithms and software to exploit emerging hardware capabilities. The dominant trend is toward increased parallelism and heterogeneous architectures, where accelerators, such as GPUs, offer the potential for new advances in computational capabilities. A coupled CFD - DL framework appears to be a natural way to harness multicore CPU/GPU systems. In this thesis, we conduct research in this direction and present new algorithms that are scalable on current and future heterogeneous systems.

Researchers use CFD in the early, conceptual design of products where it has been previously calibrated for similar applications, as well as for new configurations where little or no engineering data is available to guide design decisions. During the conceptual optimization phase, simplified models are typically used to allow reasonably accurate trades on several performance measures (for instance, the coefficient of drag, or C_D) [22, 23]. Shape design optimization in airfoils, for instance, relies on minimizing an objective function. After these early stages, CFD is also a necessary tool for a more detailed design process. For example, CFD is indispensable in designing cruise wings for commercial airplanes [24]. Finally, in the validation and certification stages of these designs, practitioners rely on CFD to answer questions that arise during product testing. Typically, the product configuration evolves over the testing period. Generally, CFD modeling capability grows to capture the required scope and physics to answer the questions raised during testing. The expense of responding to often unplanned technical surprises is a significant motivation for improving the accuracy and the speed of CFD. For each of these reasons, the performance of CFD is of critical importance.

As stated above, one of the most common approaches for aerodynamic design is shape design optimization for airfoils [25–27]. It consists of starting from a given airfoil geometry (for instance, NACA0012) and slightly modifying it to produce the desired performance values (for instance, the coefficient of lift or C_L). An example is shown in Figure 1.1. Here, we depict the original NACA0012 [28] shape together with the design variables in the optimization problem: vertical positions of the control nodes and the angle of attack α [26]. Four control nodes are in locked positions. These four variables can be modified through the optimization problem and change the original NACA0012 shape. This process lacks abstraction and generalizability because it relies on a parametrized geometry.

Alternatively, in this thesis we present a methodology that allows for a broader design space exploration using RANS methods. Even though our approach relies on high-fidelity methods

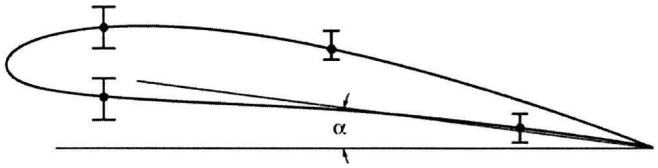


Figure 1.1: NACA0012 and its angle of attach α as an optimization variable. Also, the four locked control nodes. Common setup in airfoil design optimization [26].

rather than optimization problems, it allows us to quickly explore not only variations of one airfoil geometry but also across considerably different airfoil shapes - for instance, the ones depicted in Figure 1.2. Moreover, our method accelerates RANS simulations on airfoils from simulations on simpler geometries such as ellipses or cylinders (where the computational grid is easier to generate).

Figure 1.2: Left, the NACA6415 airfoil. Right, the NACA1412 airfoil.

We note that RANS methods have become the high-fidelity method of choice for design space exploration mostly due to the use of larger meshes, more complex geometries, and numerous runs afforded by continually decreasing hardware costs [29]. Based on feedback from the CFD survey and the follow-up workshop in [4], it is clear that the majority of the engineering and scientific community believes that RANS-based turbulence models, in conjunction with the expanded use of hybrid RANS-LES methods, will be the norm in the immediate future. Furthermore, the survey predicted the continued use of RANS with one and two-equation models, as opposed to the more complex Reynolds-Stress Transport models. Last bust not least, steady-state CFD currently accounts for the vast majority of simulations in these design explorations [4] since unsteady flow predictions are inherently

more expensive and not yet routine in the design process.

We present an in-house experiment to understand current challenges in the performance of steady-state RANS simulations. HiPer [30] is an in-house multi-block, steady-state, explicit, and compressible flow solver for structured grids. HiPer supports RANS with the one-equation Spalart-Allmaras (SA) turbulence model. HiPer incorporates several optimizations such as cache blocking, vectorization, NUMA-aware parallelization [31] and a hybrid MPI + OpenMP implementation that can run on supercomputers. HiPer also supports its own block partitioner [32]. Its large scale performance has been positively tested on several supercomputers, such as MIRA at the Argonne National laboratory and Bridges at the Pittsburgh Supercomputing Center. As acceleration techniques, HiPer includes local time-stepping, residual smoothing, blocking and parallelization (hybrid MPI+OpenMP). When simulating turbulent flow around a 3D cylinder with 33 milion computational cells, the total execution time to convergence was almost 6 hours when using 32 nodes, 24 cores per node on the Bridges supercomputer. With this baseline estimation, performing design exploration of (i) different variations of the same geometry or (ii) different flow conditions of the same geometry still becomes prohibitively expensive for having timely solutions. Moreover, immediate access to these supercomputers is not always the case because of the high demand for its compute cycles.

The cost (in convergence time) of the previous experiment comes from the inherently iterative nature of the numerical schemes. In order to meet the convergence constraints of the physics-based simulation, an approximation of the number of iterations required can range from 1×10^3 - 1×10^5 [20]. Nowadays, the most common acceleration techniques try to reduce either the execution time per iteration (parallelization, blocking) or the number of total iterations required (local time-stepping, residual smoothing, and multigrid). The previous experiment shows that CFD is still too expensive for extensive aerospace design exploration, even with the most advanced available accelerators. Since there is still significant room

for improvement, this thesis explores DL methods as a viable tool for further reducing the number of iterations and hence reducing current times-to-solution.

1.2 DL for CFD

A general approach for accelerating CFD simulations is the use of *surrogates*: mathematical models that approximate the solution of high-fidelity methods at a fraction of the computational cost. For instance, the presented RANS equations are a surrogate of DNS and are extensively used in industry for design space exploration.

However, RANS is still an iterative method. Computationally more impactful surrogates are those that approximate the solution in a single step (*one-shot* methods), reducing the computational cost significantly. Early works in this direction are, for example, regression techniques such as polynomial regressors [23, 33, 34]. However, these models have minimal predictive capabilities. Their most significant drawback is the assumed relationship (for example, linear or quadratic) between the independent variables and the predicted values, hence their inability to predict chaotic systems such as turbulent flows. To overcome this limitation, Gaussian processes (GP) predict the values that have the highest probability and a confidence interval in which these values can move - a desirable property for turbulent flows. However, GPs have a critical disadvantage: the time complexity to calibrate the parameters is $O(N^3)$, where N is the number of samples. Large datasets that cover many different flow configurations are intractable. Therefore, GPs have been used mainly in shape design optimization - for example, 2D airfoil sketch exploration - which relies on the minimization of an objective function. GPs approximate this objective function defined on a specific geometry parametrization [22]. This parametrization limits its generalizability.

Fluid mechanics has traditionally dealt with massive amounts of data from experiments,

field measurements, and large-scale numerical simulations. In the past few decades, big data have been a reality in fluid mechanics research [35] due to HPC architectures and advances in experimental measurement capabilities. Over the past 50 years, many techniques have been developed to handle such data, ranging from advanced algorithms for data processing and compression to fluid mechanics databases [36]. However, the analysis of fluid mechanics data has relied mainly on domain expertise, statistical analysis, and heuristic algorithms.

The growth of data today is widespread across scientific disciplines. Moreover, gaining information from data has become a new model of scientific inquiry. Our generation is experiencing an unprecedented confluence of (a) vast and increasing volumes of data; (b) advances in computational hardware and reduced costs for computation, data storage, and transfer; and (c) sophisticated algorithms. These advances have, in turn, fueled renewed interest and progress in the field of DL to extract information from these data. Inspired by the remarkable success of DL algorithms in both CV [37] and NLP [38], recent works have leveraged DL algorithms for accelerating CFD simulations.

DL algorithms for CFD are convenient because their inherent properties overcome the disadvantages of polynomial regressors and GP. First, DL does not assume any relationship between the input and the output, and the model can be as complex as desired by the practitioner. Second, DL training time complexity is $O(N)$, dramatically improving the training time complexity of GP. As a result, DL can model highly nonlinear and complex relationships between the input and the output if exposed to a sufficiently large number of samples while maintaining reasonable fitting times. Furthermore, recent advancements in HPC focus on increasing the number of GPUs which have constantly outperformed CPUs in training time.

Ideally, mathematical surrogates would provide a 100% accurate solution in a single shot, a highly challenging task for fluid mechanics, which presents highly nonlinear physical phenomena in a wide range of length scales and even stochastic behavior under several conditions

- where the solution is given statistically. DL, which achieves deep, abstract mathematical relationships between input and output, poses a potential candidate for bringing researchers one step closer to the ideal scenario described before. Hence, it yields a natural research direction: using DL to predict expensive CFD results and trust its prediction.

Several researchers have leveraged neural networks (NN) to accelerate fluid dynamics simulations in the past few years. Tompson et al. [39] 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. Alternatively, Guo et al. [40] find a real-time solution to viscous laminar flows around solid objects. However, the above approaches have elemental constraints. First, Eulerian fluid simulations ignore second-order velocity derivatives in the NS equations. Second, viscous laminar flow approaches are ambiguous to expand to turbulent flows - intrinsically chaotic and harder to resolve [1].

There have been recent attempts to find NN-based accelerators for turbulent flows. The results are promising, but the NN predicts only a subset of the flow variables. Maulik et al. [41] predict the eddy viscosity field and not other flow properties such as velocity and pressure fields. Thuerey et al. [42] use a novel input-output representation, but their approach does not account for the eddy viscosity field. More importantly, the network prediction is limited to the fluid domain closest to the solid body, so how the network would perform in the freestream - where the boundary conditions define the problem - remains unknown.

A fundamental limitation of these early works [40, 42] is that they use the NN as a data-only, end-to-end surrogate. Therefore, they can not guarantee that the NN prediction will satisfy the conservation laws because the conservation laws were not part of the optimization process. Satisfying the conservation laws of the problem at hand is critical for practitioners. Moreover, even though these works report promising results at test time - a relative mean error (RME) less than 3% [42] - 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.

Another limitation of the approaches discussed so far is that they require data collection for training the NN. Often, this data comes from traditional fluid simulations. Hence, these approaches predict a solution in a domain discretized at the exact resolution used to collect the data. These approaches target low resolutions, which are insufficient in real-life aerospace design scenarios. How do we scale these methods to larger resolutions that are the preferred choice of designers in the aerospace industry? Performing data collection at high resolutions, such as 2048×2048, is impractical. Early DL-based *mesh-free* methods arose as an alternative to circumvent this limitation.

Raissi et al. [43] introduced *physics-informed neural networks* (PINNs) - NNs embed the residual of any physical law in the loss function and respect any physical constraint. PINNs [43–45] are a promising research direction because they are an end-to-end, physically-consistent substitute to traditional solvers[2] and do not require tedious mesh generation because they are *mesh-free* methods. However, this approach comes with challenges. First, for complex turbulent flow problems, including the conservation laws in the loss function can lead to stiffer optimizations [47]. Therefore, the majority of the discussed approaches target low to moderate Reynolds numbers (Re)[3] [46]. Second, unless the boundary conditions are input to the network [48], training a new model is required for every new instance of a distinct flow configuration, instead of a simple forward pass of the network, severely restricting its generalization capabilities. Third, even if the boundary conditions are input to the network, generalizing to significantly different-from-training geometries/domains is challenging and unexplored in the majority of approaches. Fourth, although the mesh generation step is avoided, it remains an open question how many collocation points this method requires for

[2]PINNs are already impactful in industry. NVIDIA's SimNet solver [46] can query multiple instances of a heat transfer problem in real-time.

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

accurate solutions of the problems. Finally, there is still a gap of several orders of magnitude between the residual value achieved by traditional solvers and PINN-based surrogates. While traditional CFD solvers such as OpenFOAM can reach residual values up to 1×10^{-8} (the lower, the better) and satisfy physical laws up to the machine round-off errors, the majority of PINN-based residuals struggle to drop below 1×10^{-3} or 1×10^{-4}, even for less complex problems.

Lu et al. [49] introduced a *mesh-free* infinite-dimensional operator with NNs, known as *neural operator* (NO), that learns the nonlinear operation from PDEs without knowledge of the underlying PDE – only with data. NO provides a single set of network parameters compatible with different discretizations. Hence, they are resolution-invariant. However, these approaches [50–52] train the network with data downsampled from high-resolution simulations – which is impractical for many practitioners and suffers from the same computational constraints of traditional solvers. Jiang et al. [53] introduced *MeshFreeFlowNet*, a convolutional neural network (CNN) based resolution-invariant approach that satisfies the underlying PDE. However, it also suffers from the same data-collection limitation as the former approaches. In this thesis, we present a transfer learning-based approach that uses limited data collection to scale our DL-based accelerator to high spatial resolutions - up to 2048×2048, significantly larger than those attempted in prior works (421×421 [50]; 512×128 [53]; 1024×1024 [54]).

One fundamental limitation of all the approaches discussed so far is that they perform *uniform* super-resolution; that is, they try to predict a high-resolution flow field throughout the entire domain. Hence, they soon reach the given hardware limits during inference of high spatial resolution outputs, such as 2048×2048. However, very-large outputs might not be necessary. In CFD, a popular approach to scale to very large resolutions while maintaining accuracy is adaptive mesh refinement (AMR) [55]. AMR augments the resolution only in areas that require higher numerical accuracy (such as those close to the solid body)

while maintaining low resolution in regions that present a smoother behavior (such as the freestream). Unfortunately, state-of-the-art (SOTA) DL methods for CFD do not attempt AMR when it could be beneficial given limited hardware resources. This thesis presents a DL-based method for AMR that allows larger batch sizes during inference at high spatial resolutions and significantly speeds up DL-based prediction of high-resolution flow fields.

1.3 Research approach and contributions

The CFD community finds itself currently at a stagnation point. First, from the HPC standpoint, clusters are becoming heterogeneous (CPU + GPU), and even when trends are hard to predict, this heterogeneity is expected in the next 5 to 10 years. Current algorithms for CFD are old and not adapted to these new hardware systems. Therefore, there is a need to adapt CFD algorithms to current hardware capabilities. Second, there is still room for improvement in times-to-solution of current CFD solvers. Due to the impressive results of DL methods in CV and NLP, but also in early works that predict complex fluid problems in real-time, we believe DL can further accelerate these systems and provide practitioners additional tools to speedup high-fidelity aerospace design exploration.

- **Contribution 1. A DL-based accelerator that can be used on top of current acceleration techniques**. We present CFDNet, a DL framework that combines domain-specific knowledge to meet the same convergence constraints of the physical solver. CFDNet presents an *iterative refinement* stage, where the CNN output is fed back as the initial condition to the physics solver. Therefore, CFDNet satisfies the conservation laws of physics solvers while accelerating them. 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 (for instance, channel flow); training on multiple geometries (ellipses) and predicting on a subset (for

instance, an 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 (for instance, airfoil or cylinder), which is challenging. CFDNet exhibits remarkable generalizability to geometries unseen during training while achieving $1.9 - 7.4\times$ speedup on laminar and turbulent flows over a widely used physics solver in OpenFOAM. CFDNet is a DL module amenable to any time-marching scheme and discretized domain. Furthermore, CFDNet can be used on top of local time-stepping, blocking, multigrid, and parallelization - the most popular acceleration techniques in CFD.

CFDNet's experiments confirm it as a viable proof of concept for low-resolution CFD simulations. However, it relies on extensive data collection, and hence it becomes computationally prohibitive for high-resolution simulations. To avoid the need for extensive data collection and training at high resolutions and to overcome the already exposed limitations of *mesh-free* methods, we present SURFNet, a transfer learning-based framework. SURFNet scales CFDNet to high resolutions with limited high-resolution data collection.

- **Contribution 2. A transfer learning-based framework for scaling DL methods for CFD to high resolutions**. We develop SURFNet (**SU**per-**R**esolution **F**low **Net**work), a novel approach to reconstruct fine-scale flow physics from coarse grid data by primarily training the DL model on low-resolution inputs. SURFNet transfer learns this coarse model to high-resolution turbulent flow solutions, significantly reducing the overall data collection time and the total size of the training set. We empirically evaluate SURFNet by solving the RANS equations in the turbulent regime on four geometries and eight flow configurations unseen during training. SURFNet achieves a consistent $2\times$ speedup up to 2048×2048 spatial resolutions over the OpenFOAM physics solver independent of the resolution size and test geometry, demonstrating both resolution-invariance and generalization capabilities. SURFNet eliminates the

need to collect complete training datasets at high resolutions to account for fine-scale physical phenomena. This computational efficiency enables SURFNet to achieve oracle accuracies while significantly reducing the size of the training dataset by 15×, consequently reducing the combined data collection and training time by 3.6× and 10.2×, respectively at 256×256 and 512×512 grid sizes.

SURFNet is a time-efficient framework for the acceleration of high-resolution CFD simulations. However, it does not entirey eliminate the small but tedious data collection at high resolutions. Moreover, SURFNet reaches the hardware limits - overallocation of memory and big inference times - when performing inference at 2048×2048. To reduce the hardware requirements but still scale to very-large resolutions, we present ADARNet, a novel, DL-based method for AMR, which uses a novel *scorer-ranker* NN architecture to increase the resolution of some areas of the domain - those that require higher accuracy.

- **Contribution 3. A DL-based framework for non-uniform super-resolution**. We present ADARNet, a novel DL-based **ADA**ptive mesh **R**efinement framework for non-uniform super-resolution. ADARNet takes as input a low-resolution flow field and outputs, in one single step, its final non-uniform high-resolution solution. Since ADARNet only increases the resolution in areas that present complex flow phenomena, it requires less computational resources. It allows larger batch sizes during inference at high spatial resolutions while reaching the target accuracy compared to SOTA methods for super-resolution. To enable non-uniform super-resolution, we split the input image in different regions or *patches* and present a novel scorer-ranker-decoder DL algorithm. The *scorer* finds the spatial score of each patch, the *ranker* places each patch in its corresponding bin based on its score (which determines the target resolution of the patch), and the *decoder* reconstructs every patch in each bin to its final target resolution using semi-supervised learning. Due to ADARNet's ability to only refine specific areas of the flow and avoid high-resolution inferences in most parts of the domain,

it achieves a speedup of $7 - 28.5\times$ and reduces the memory usage by $4.4 - 7.7\times$ at 1024×1024 spatial resolutions compared to SOTA DL methods that perform uniform super-resolution while reaching SOTA accuracies.

We organize the rest of this thesis as follows. Chapter 2 presents the necessary CFD and DL background. Chapters 3, 4, and 5 present, respectively, CFDNet, SURFNet, and ADARNet, discussing their advantages and disadvantages. Finally, we make concluding remarks in Chapter 6, along with a discussion of future research directions.

Chapter 2

Background

In this chapter, we introduce the necessary CFD and DL concepts. In Section 2.1, we present the RANS equations together with the SA model. In Section 2.2 we present an overview of DL principles for real-time inference of flow fields.

2.1 RANS equations and SA model

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

$$\frac{\partial \bar{U}_i}{\partial x_i} = 0 \tag{2.1}$$

$$\bar{U}_j \frac{\partial \bar{U}_i}{\partial x_j} = \frac{\partial}{\partial x_j}\left[-\left(\bar{p}\right)\delta_{ij} + \left(\nu + \nu_t\right)\left(\frac{\partial \bar{U}_i}{\partial x_j} + \frac{\partial \bar{U}_j}{\partial x_i}\right)\right] \tag{2.2}$$

where $\bar{U}$ is the mean velocity (a 2D or 3D vector field), $\bar{p}$ is the kinematic mean pressure, ν

is the fluid viscosity, and ν_t is the eddy viscosity resulting from Boussinesq's approximation [56]. Typically, turbulence modeling is used for the eddy viscosity (ν_t). The SA one-equation model shown below provides a single transport equation to compute a modified eddy viscosity, $\tilde{\nu}$.

$$\bar{U}_i \frac{\partial \tilde{\nu}}{\partial x_i} = C_{b1}\left(1 - f_{t2}\right)\tilde{S}\tilde{\nu} - \left[C_{w1} f_w - \frac{C_{b1}}{\kappa^2} f_{t2}\right]\left(\frac{\tilde{\nu}}{d}^2\right) + \frac{1}{\sigma}\left[\frac{\partial}{\partial x_i}\left(\left(\nu + \tilde{\nu}\right)\frac{\partial \tilde{\nu}}{\partial x_i}\right) + C_{b2}\frac{\partial \tilde{\nu}}{\partial x_j}\frac{\partial \tilde{\nu}}{\partial x_j}\right] \quad (2.3)$$

Then, we can compute the eddy viscosity from $\tilde{\nu}$ as $\nu_t = \tilde{\nu} f_{v1}$. These equations represent the most commonly-used implementation of the SA 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 [14], the first original reference of the model.

Equations (2.1), (2.2), and (2.3) form a system of four PDEs in 2D and five PDEs in 3D. Throughout this thesis, we numerically solve the discretized form of these equations on a structured grid with its corresponding boundary conditions (that define the physical boundaries). The spatial 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, unbounded scheme. Finally, the diffusion terms in Equation (2.2) and Equation (2.3) 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.

2.2 DL overview

DL algorithms are a subset of machine learning (ML) algorithms because they are mathematical models that try to fit to the data that is presented to them. The main difference between DL and other ML algorithms is that it makes use of NN (similar to neurons present in human brain) to imitate functionality just like a human brain. The reason why DL algorithms can become very complex is because they are based on the concept of a perceptron or *neuron*, as depicted in Figure 2.1.

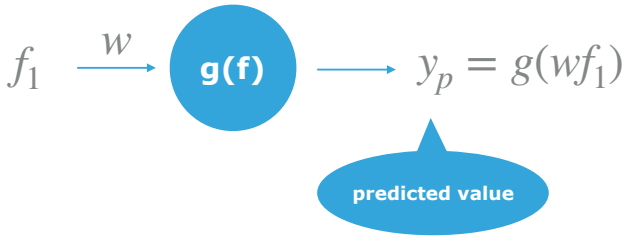


Figure 2.1: Perceptron, input, output, and activation function.

A perceptron takes an input feature f_1, it weights the value of the feature with w, and it passes it through an activation function $g(f)$. To sum up, a perceptron outputs a transformation of a weighted value of the input feature. For example: let's imagine that f_1 is the one-dimensional constant acceleration (a) of a car for a period of time. Let's also assume $g(f) = 1$. Then, the output from the perceptron would be $y_p = wf_1 = wa$. DL algorithms fit their parameters (the weights, in particular, w) to the given data. **The goal of the *training* or *learning* task of a perceptron is to find the values of w that correctly predict the given, ground truth data presented to this perceptron**. Therefore the

prediction y_p must be compared to ground truth data:

$$y_p - y_t = 0 \tag{2.4}$$

where y_t is the ground truth output. In the previous example, we choose y_t to be the total one-dimensional force measured on the car, F_t:

$$y_p - y_t = wa - F_t = 0 \tag{2.5}$$

After training, it would be revealed that $w = m$ where m is the mass of the car (Newton's second law). Now, this calibrated model can predict F_t on the car for any constant acceleration that the car has, because the perceptron has learned $w = m$.

This simple example does not take into account that $g(f)$ can be a non-linear function (for example, the hyperbolic tangent). Also, it does not take into account that two perceptrons can be aligned forming a more complex output, creating more than one layer of perceptrons:

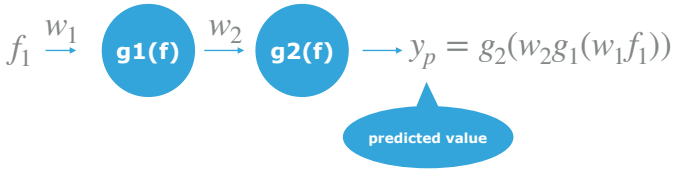


Figure 2.2: Two layers of a single perceptron. y_p is the output, a complex, non-linear transformation of the input feature f_1. g_1 is the first activation function and g_2 is the second activation function.

Figure 2.2 shows that the predicted value can be a very complex transformation of the input feature. Also, there might be more than one feature relevant to the mathematical model as shown in Figure 2.3

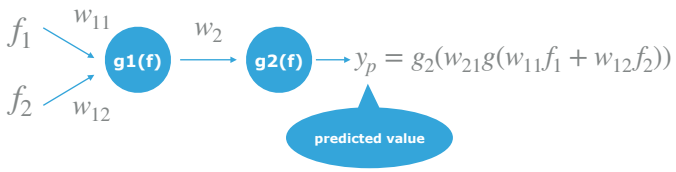


Figure 2.3: Two features as input to two layers of a single perceptron. y_p is the output, a complex, non-linear transformation of two input features, f_1 and f_2. g_1 is the first activation function and g_2 is the second activation function.

In general, this perceptron layout allows to build complex nonlinearities of multiple features. If we add as many features as we want, and distribute as many perceptrons or *neurons* as desired, it is possible to create a deep NN, as depicted in Figure 2.4.

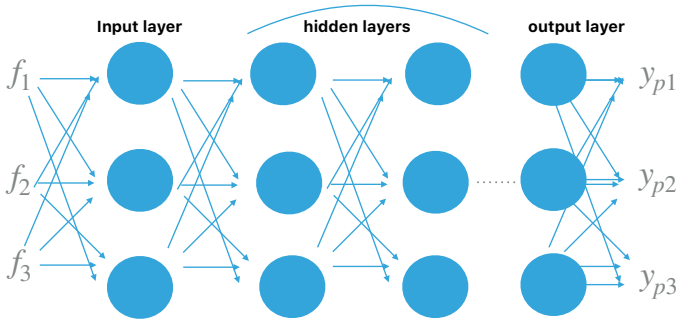


Figure 2.4: A basic DNN with several input features. The input layer receives the input features and performs the first transformation. The hidden layers extract abstract patterns from the input data, and there can be as many as desired by the practitioner. The final layer, namely the output layer, outputs the target value or transformed feature.

To sum up, deep neural networks (DNN) allow (a) high-dimensional inputs (as many features as desired by the practitioner), (b) complex, non-linear relations among these features, and (c) extensive hyperparameter space exploration for finding the suitable network complexity (how many neurons per layers? How many layers? What is $g_n(f)$?). So, if sufficient ground truth data is presented to a DNN, it can become a complex mathematical model. This is the main advantage of DNNs over traditional ML algorithms and why DL outperforms ML algorithms for complex problems (for instance, image classification).

DL refers to the stage of calibrating the weights of a DNN. This calibration is done by comparing the network's output (y_p) to the ground truth data (y_t). However, this is one of the two options for weight calibration. This process is called *supervised learning* [57]. It is *supervised* because the user tells the network what the ground truth data (for example, *the force must be* 3 N) is and the prediction of the network needs to approximate it. An alternative to this approach is not comparing the network's output to any value. Instead, the user knows that the network's output needs to satisfy some property (for example, the user imposes the output to be always 0). In this case, the network's output does not need to be compared to any labeled data. In this case, the approach is called *unsupervised learning* [58]

CNN. CNNs are a subset of DNNs: a CNN is a DNN where at least one is a convolutional layer. We define the order of the features (f_1, f_2 and f_3 in Figure 2.4) as follows: the features f_1, f_2 and f_3 are given as a feature vector, $\vec{f} = [f_1, f_2, f_3]^T$. This vector could also be built as $\vec{f} = [f_2, f_1, f_3]^T$. These two vectors have a different order of the features.

This definition is important because it plays a crucial role in differentiating a CNN from a DNN. As it can be seen in Figure 2.4, each perceptron of the initial layer is connected to all features, with its corresponding weights. Therefore, the perceptron is blind to the order of the features. It does not care if $\vec{f} = [f_1, f_2, f_3]^T$ or $\vec{f} = [f_2, f_1, f_3]^T$, because each perceptron will have an output such as $y_p = g\left(\vec{w} \cdot \vec{f}\right) = g(w_1 f_1 + w_2 f_2 + w_3 f_3)$. This is a property of *dense* or *fully connected* layers. Therefore, Figure 2.4 is a *fully connected* DNN.

CNNs, instead, have at least one non-fully connected layer: a convolutional layer. Here, the order of the features is critical. Two different orders will lead to two different outputs or y_p. Because the order of the features is important, CNNs work very well on 2D/3D or matrix-like (image-like) input features. Figure 2.5 shows the operator of a convolutional layer, which is different from the fully connected perceptron $g\left(\vec{w} \cdot \vec{f}\right)$.

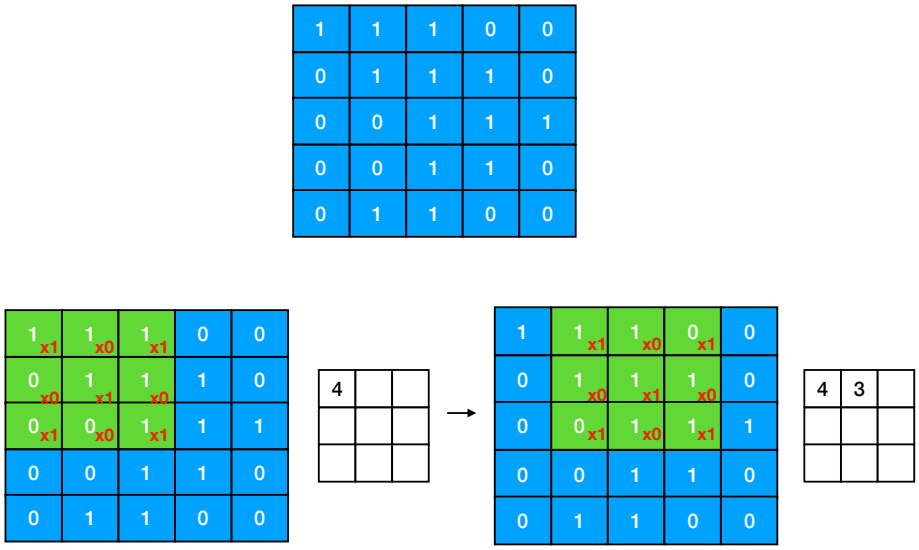


Figure 2.5: The convolving operator. Top: the input 5×5 image with the value at each pixel. Left: applying the 3×3 kernel (in red) on the top-left 3×3 area of the input image (in green). Right: moving the kernel to cover all 3×3 regions of the input image.

The input features can be displayed as a matrix instead of being displayed as a vector ($\vec{f} = [f_2, f_1, f_3]^T$). In Figure 2.5, that is the blue matrix (or image). Each element of the matrix is a feature. Let us imagine that the blue matrix is an image of a cat. Each element of this matrix is the pixel, and the value is the corresponding value of the color (0-255 in RGB). A convolutional layer convolves these features using a filter of a certain size. The size chosen in the Figure is a 3×3 filter. The red numbers are the values of the weights in that filter, which need to be calibrated. The filter acts on a 3×3 portion of the input matrix. The filter convolves those nine values and outputs the number 4. Then, the filter moves one column to the right and does the same for the following 3×3 set of features. Therefore, a convolutional layer is excellent at finding correlations in certain parts of the input matrix. If the convolutional layer observes this correlation in different training samples, it can recognize patterns. As a result, CNNs are ideal candidates for edge detection. This could also be achieved with a fully connected NN; however, the CNN shares the weights in different image locations, whereas a fully connected layer would have independent weights at each location.

The patterns could be found, but it would take longer training times. CNNs have a spatial inductive bias in their architecture. As a result, CNNs have shown outstanding performance in image classification, NLP, and image and video recognition [6, 7].

Chapter 3

CFDNet: A new acceleration technique

To find the steady-state solution to equations (2.1), (2.2), and (2.3), it is necessary to set the required boundary conditions and numerically solve them until convergence. For that purpose, we use the semi-implicit method for pressure-linked equations (SIMPLE) algorithm [59] outlined in Algorithm 1.

The SIMPLE algorithm is a well-known iterative procedure widely used in literature [12, 60, 61] 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 (that is, 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 research is to *short-circuit* the convergence progress by reducing the number of iterations of the RANS simulation. For the rest of this Chapter and Chapter 4, Algorithm 1 will be referred as the *physics solver*.

The goal 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

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 (2.2)
3: Solve $\bar{p}^{'}$, a correction value for $\bar{p}$, from Equation (2.1)
4: Obtain correct $\bar{U}$ and $\bar{p}$ fields from $\bar{p}^*$, $\bar{U}^*$ and $\bar{p}^{'}$
5: Solve ν_t from Equation (2.3)
6: **if** residual $< \epsilon$ **then**
7: **return** done
8: **else**
9: $\bar{p}^* \leftarrow \bar{p}$
10: **goto** 2

in Algorithm 1, and I $(0 \leq I \leq N-1)$ represent any intermediate iteration of the solver. The aim is to create a map such that $x^N = G\left(x^I; \theta_{wl}\right)$ 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.

Figure 3.1 displays a visual representation of G. Usually, the user is only interested in the final iteration of a steady-state simulation, which provides the converged steady-state solution. The intermediate iterations, even though required by the numerical scheme to reach the final steady-state, are unnecessary/useless from the user's point of view. The goal, therefore, is to skip a significant number of iterations by mapping the flow field at any intermediate iteration - the earlier the better - to the final steady-state flow field.

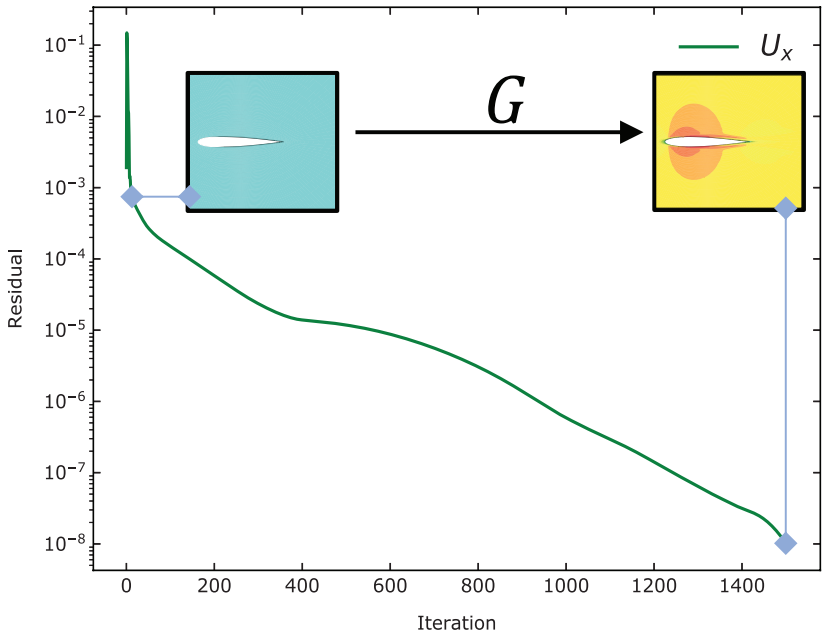


Figure 3.1: x-velocity residual for flow over an airfoil at $Re = 6 \times 10^5$. G is a function that maps any intermediate iteration to the final steady-state solution.

The expectation is that if the coefficients of G - θ_{wl} - are calibrated with simulations of several specific flow configurations, G can become generalizable and capable of accelerating the simulation of a flow configuration not previously seen by G by finding its steady-state solution from an early intermediate iteration. This approach allows to use the flow field at intermediate iterations regardless of how they have been computed. For example, the simulation in Figure 3.1 could be run using parallelization and multigrid as acceleration techniques, and the approach would still be valid. Finding G provides a new acceleration technique that can be used on top of the most successful and current ones. The approach is also independent of the iterative scheme used. Even though in this research the SIMPLE algorithm is assumed to be the physics solver, this approach is generalizable to any time-marching scheme, such as explicit Range-Kutta schemes [62].

A DNN, specifically a CNN is the candidate chosen for G (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. Let's now assume that the CNN has calibrated its weights θ_{wl} through a *learning* task with different finished simulations of several flow configurations and recall that the CNN is trained to predict the velocity, pressure and eddy viscosity field in the entire fluid domain at steady-state. Therefore, now it is ready to be integrated in a simulation of a new flow configuration. The aim is that from an intermediate iteration of this new simulation, the CNN is able to predict the expected steady-state solution of this particular new flow configuration in *one single inference step*, thus dramatically reducing the number of iterations required. A common approach would be to consider these CNN-inferred fields as the final solution to the problem if the CNN has good generalization properties. This approach is the methodology considered in most of the SOTA approaches outlined in Section 1.2.

However, in this work, the output from the CNN is fed back into the physics solver as a new initial condition, from which the residuals are dropped. This is what we refer as *short-circuiting* the convergence process: we expect that the output of the CNN is numerically closer to the CFD-based steady-state solution so that only few iterations are required to reach the convergence constraints. This framework, referred as CFDNet, is detailed in this chapter, from its design to its performance on several case studies.

3.1 CFDNet

CFDNet is a DL framework that combines domain-specific knowledge to meet the same convergence constraints of the physics solver. The objective with CFDNet is to produce the same output as the physics solver (that is, respect the convergence constraints) while accelerating the convergence process using DL. CFDNet is a *physics solver-CNN model-physics solver* coupled framework instead of a pure surrogate (that is, considering the output of the CNN as the final solution). We designed it this way to incorporate domain-specific knowledge in the model inference and to reach the same convergence as the physics solver.

Figure 3.2 depicts the CFDNet framework and below is the description of the three main stages of the framework and the two key ways domain-knowledge is integrated.

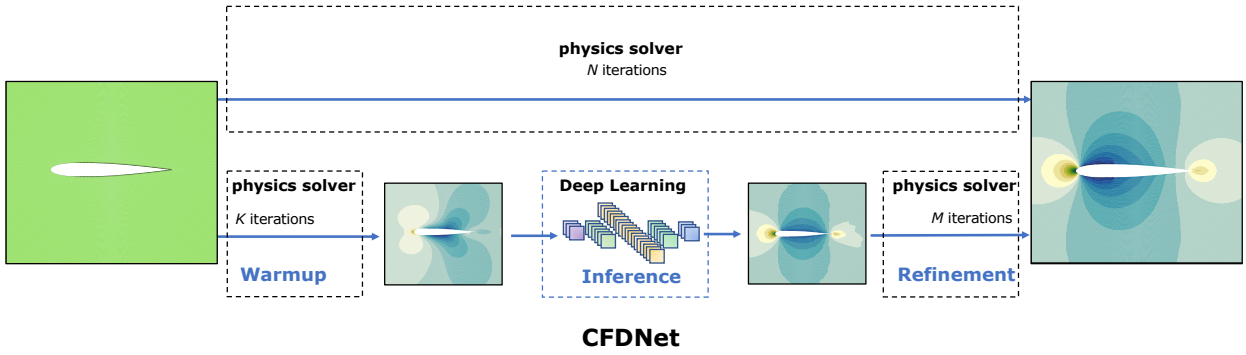


Figure 3.2: Comparison of the traditional physics solver simulation with CFDNet. CFDNet integrates the domain-specific physics solver for *warmup*, followed by the NN for inferring the steady state, and the final iterative *refinement* stage to correct the solution of the CNN and satisfy the convergence constraints.

1. ***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 (2.1), (2.2), and (2.3). K is determined 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. This is the warmup stopping criteria.

2. ***CNN Inference***. After warmup, we generate the input to the network, which is an image (a tensor) containing the values of the flow variables at iteration K (the generation of the this input image is detailed in the next sections). 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.

3. ***Iterative Refinement***. The output of CNN inference may not satisfy known conser-

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

3.1.1 How CFDNet improves SOTA approaches

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, DL algorithms have demonstrated remarkable improvements in a variety of modeling/classification tasks such as NLP, CV, and HPC [37, 38, 63]. Several researchers have applied DL algorithms to accelerate CFD simulations. Table 3.1 summarizes the SOTA methods by comparing them on nine different features.

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. [39]	●	●	●	●	●	●	●	●	●[1]
Smart-fluidnet [54]	●	●	●	●	●	●	●	●	●[2]
Guo et al. [40]	●	●	●	●	●	●	●	●	●[3]
TBNN [5]	●	●	●	●	●	●	●	●	●[4]
Maulik et al. [41]	●	●	●	●	●	●	●	●	●
Zhu et al. [64]	●	●	●	●	●	●	●	●	●[5]
Thuerey et al. [42]	●	●	●	●	●	●	●	●	●[6]

Table 3.1: SOTA methods in CFD acceleration using deep learning on nine features. [(1)(2)(4)(5)] replace particular steps of the original iterative algorithm with a DNN (find a surrogate of the Poisson solver[(1)(2)], estimate the Reynolds stresses[(4)], estimate the eddy viscosity[(5)]), and [(3)(6)] replace the entire algorithm with a CNN-based surrogate to estimate the final solution of the fluid variables of interest.

While the current approaches have shown the feasibility and potential of DL 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 NN predicts only a subset of the flow variables which provide incomplete information about the problem. Others do not 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 DL is challenging especially for unseen geometries. Overall, we observe that SOTA approaches satisfy up to six out of the nine features shown in Table 3.1.

CFDNet tackles these limitations and makes the following contributions:

1. One single CNN architecture - as explained later - is used to predict *all* relevant flow variables, that is, velocity in each direction, pressure and eddy viscosity of the fluid. CFDNet is the first DL and physics coupled framework to predict all relevant fluid variables in the entire fluid domain of a turbulent flow.

2. The iterative refinement stage takes the output from the CNN and feeds it 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 NN is modeled as an end-to-end surrogate and there is no guarantee that conservation laws are satisfied.

3. Several use-cases are considered in the evaluation of the accuracy, performance, and generalizability of CFDNet as an accelerator. These include training and prediction on the same geometry (for example, channel flow), training on multiple geometries (ellipses) and predicting on a subset (for example, 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 (for example, airfoil or cylinder) which is challenging.

3.1.2 Input/Output representation

This section discusses the input/output representation (that is, how the tensors x^I and x^N are created, the input and output to the network).

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 ($\tilde{\nu}$ [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, to generate the input-output representation in Figure 3.3 as follows:

1. Compose an image of size $m \times n$, where $m \times n$ is the grid size. Every pixel (i, j) of this image has the corresponding 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, augment the rows of this image by appending the physical boundary values of the variables from the computational grid (that is, the values at the top and bottom boundaries) leading to an image size of $(m + 2) \times n$.

2. Repeat the previous step for the other flow variables of interest – $\bar{v}$, $\bar{p}$ and $\tilde{\nu}$, leading to four images in total, each of size $(m + 2) \times n$.

3. Concatenate the previously generated four images to create the final input/output representation to the CNN which is a tensor of size $(m + 2) \times n \times 4$ (that is, an image

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

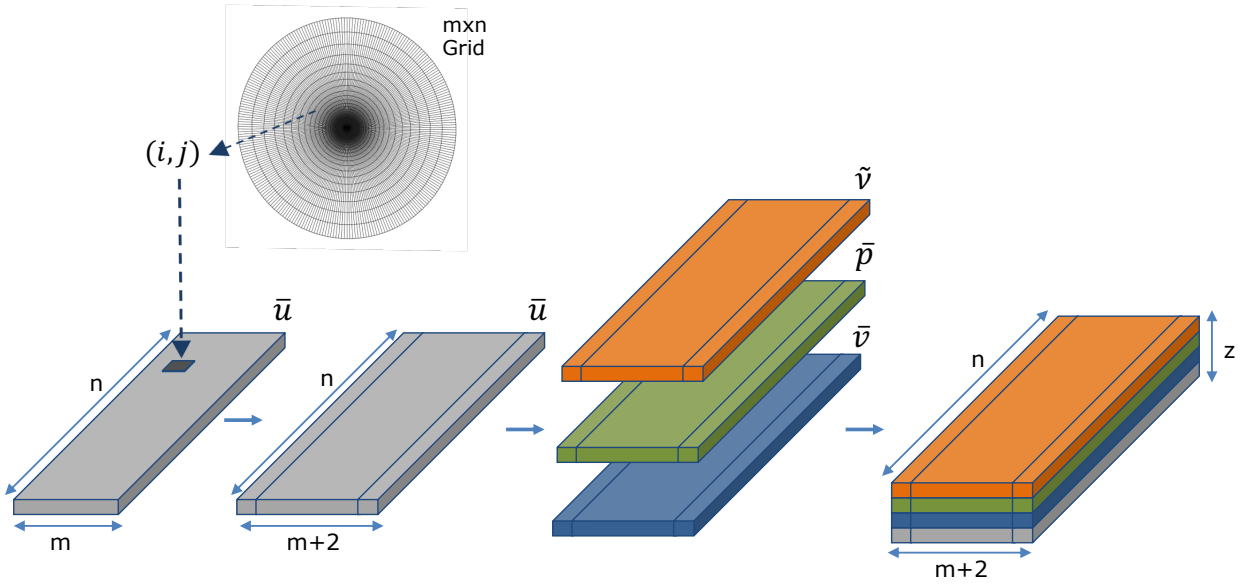


Figure 3.3: Input-Output Representation in CFDNet. The input and the output are images of size $(m+2) \times n \times z$ because $m \times n$ is the grid size, 2 are the top and bottom boundaries, and z is the number of flow variables (channels). Input image has the grid values of the variables at an intermediate iteration and the output image has the values in the final steady solution.

consisting of four channels).

Both input and output 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, which is conducted 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 $\mathrm{m\,s^{-1}}$ for velocity and 1×10^{-6} to 1×10^{-4} $\mathrm{m^2/s}$ 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-dimensionless parameters are significantly smaller than others, they are negligible in certain areas of the flow [65].

3.1.3 Network design

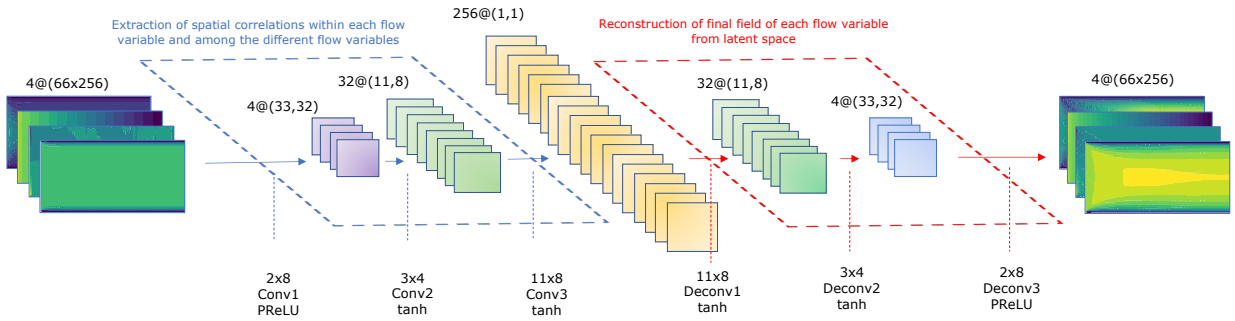


Figure 3.4: CNN architecture. The CNN is a symmetric 6-layer CNN. The first three layers are Convolution layers and they are followed by three Transposed Convolution layers. The size of each filter is outlined in the figure and a striding of the same size as the filter is applied in each layer.

A CNN is the candidate to learn the *surrogate* model for accelerating the RANS equations. This network is composed of six layers, three convolution and three deconvolution layers[2]. 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 [66]) 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. The output is a prediction of *all* the steady-state flow variables ($\bar{u}$, $\bar{v}$, $\bar{p}$, and $\tilde{\nu}$) as shown in Figure 4.2.

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 [67, 68]. Second, the spatial layout of the flow variables in the input tensor is a result of Equations (2.1), (2.2), and (2.3). The convolution operator is the ideal candidate to extract the existing spatial correlations in and among the fluid variables (that is, input channels), while fully connected layers fail to do so.

[2]More expressive networks (8- and 10-layer CNN) led to overfitting to the training datasets.

There is no explicit *domain-specific* feature in the network's input and output because the aim is to learn a model that generalizes to a broad number of design use cases comprising of both different - interpolated and extrapolated - geometries and flow conditions. Prior works that embed domain-knowledge in the NN *learning* task (such as embedding Equation (2.1) as the loss function in [39]) fail to achieve this generalization limiting its scope.

3.1.4 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 (that is, 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 SA 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 [69]. 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 the CNN model as well). Even though this optimization choice makes the model more generalizable (for example, 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 [39, 54, 64] 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 [54]. 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.

Circumventing the above limitations is achieved with the *refinement* stage in CFDNet and define its *dual convergence criteria* as follows.

1. Convergence is typically achieved when the residual of all the variables has dropped 4-5 orders of magnitude [5] depending on the CFD practitioner. CFDNet is adhered to the same constraints. Because the residual is referenced to the initial condition, and the 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 CFD engineers to also ensure that the final solution satisfies conservation properties such as *conservation of mass*. The author adheres to this convergence constraint and this is the second convergence criterion of CFDNet.

When both *residual convergence* and *conservation laws* are satisfied, CFDNet satisfies the *convergence constraints* of the original physics solver and has a solution with 0% 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.

3.2 Experiments

Two types of fluid flows are considered - wall-bounded flows and external aerodynamics - which aim to stress different aspects of CFDNet:

- For the former, the geometry is kept constant while changing the flow configurations. This tests the ability of CFDNet to interpolate and extrapolate to new flow conditions in the same geometry.
- For the latter, the flow conditions are kept constant while changing the geometry. Here, we evaluate the ability of CFDNet to acclerate simulations of flows around solid bodies not previously seen by the CNN.

3.2.1 Case studies

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 (for example, pipe and channel designs). The study of these flow configurations is relevant for a broad range of Re 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(given that $Re = \frac{UL}{\nu}$, where U is the input velocity to the channel, L is the diameter of the channel, and ν is the laminar viscosity of the problem).

In channel flow, transitional effects from laminar to turbulent flow occur around $Re \approx 3000$ [1]. Therefore, $Re = 4000$ is the lower bound in the 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, the range scoped is 4000 < Re <15000 for the design exploration. The CNN model is trained 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, Re = 5600 and Re = 13750.

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, CFDNet is applied 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 3.5. 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 6×10^5 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).

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 3.6.

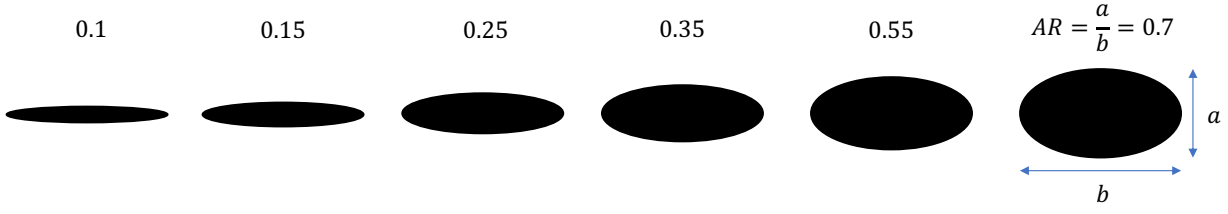


Figure 3.5: Ellipse geometries used for training in datasets B and C.

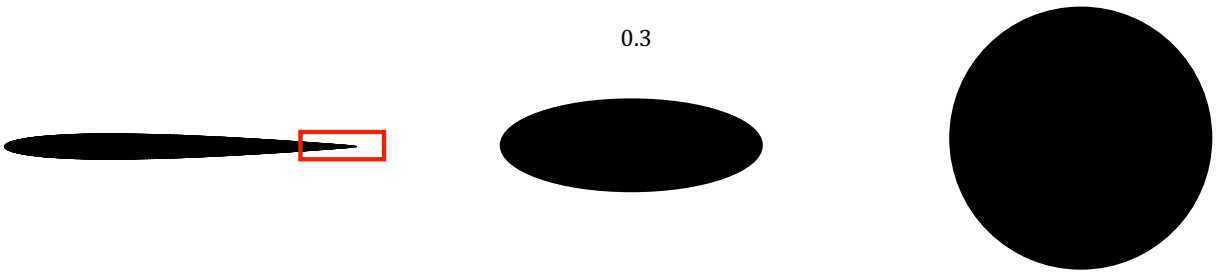


Figure 3.6: Airfoil, ellipse, and cylinder used for testing the CFDNet framework. Highlighted is the trailing edge of the airfoil as a new edge not seen by the network in training.

The physics solver in OpenFOAM: The training dataset is generated by solving the RANS equations together with the SA one-equation model [14]. The grid is 64×256, a proper grid dimensionality for all cases [70]. We use the incompressible solver `simpleFoam` from OpenFOAM v6 as the *physics solver* in this work. The simulations are run till steady state reaching a residual value of 1×10^{-6} for the velocity and pressure and 1×10^{-4} for the modified eddy viscosity in the channel flow cases. For the flows around solid bodies, a residual value between 1×10^{-5} and 1×10^{-6} for the velocity and pressure and 1×10^{-4} for the modified eddy viscosity is considered acceptable. These tolerances are extended practice in CFD simulations [5].

Note that the 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 implicitly computing the pressure at every iteration, with a tolerance 1×10^{-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.

3.2.2 Dataset generation

Three distinct datasets are created based on the two case studies. 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. Perform the OpenFOAM simulation of all training flow configurations. During the simulation, snapshots of the flow field at every intermediate iteration are taken, 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 the inputs to CFDNet as detailed previously. 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 (that is, 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 (for example, $n_f = 4$ for dataset A). Now, the initial conditions (iteration 0) are also added as a sample to the dataset and the training set size then becomes $\sum_{j=1}^{n_f} N_j$ for each dataset.

2. *Non-dimensionalize* all the samples. The fluid variables of all images are non-dimensionalized

as follows: $\frac{\bar{u}}{U_r}$, $\frac{\bar{v}}{U_r}$, $\frac{\bar{p}}{U_r^2}$ and $\frac{\tilde{\nu}}{\nu_r}$, 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.

3.2.3 Training

Three independent models are trained, one for each dataset A, B, and C. The CNN (shown in Figure 4.2) is implemented using Keras [71] and training of the CNN is performed on a Tesla K40c NVIDIA GPU using the TensorFlow 13.0 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 7×10^{-5} with no decay for all training. For training set A, 15 epochs were sufficient to drop the training loss and validation loss to 3.2×10^{-6}. For training set B, after 35 epochs the training loss reached 1×10^{-4} and the validation loss 2×10^{-4}. For training set C, the training and validation losses dropped to 2×10^{-5} and 5.1×10^{-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.

3.3 Results and discussion

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

Observed geometry, different flow conditions (OG-DF). CFDNet aims to accelerate the simulation of a flow on a geometry observed during training but on different flow configurations. Here, the flow case is channel flow and the CNN model is trained with dataset A. Two different flow conditions are tested – an input velocity to the channel [3] of 0.56 $\mathrm{m\,s^{-1}}$ and 1.375 $\mathrm{m\,s^{-1}}$. With the former, we evaluate CFDNet's capacity to interpolate to a new flow condition, whereas with the latter, we evaluate its ability to extrapolate.

Subset geometry, same flow conditions (SG-SF). The test geometry is a *subset* of the training dataset. Here, the aim is 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 (that is, interpolation). In this use-case, the flow conditions are kept the same throughout the experiment. Here, the 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 3.5 and the test geometry is also an ellipse (with an

[3]Note 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).

unseen-in-training AR) shown in Figure 3.6.

Different geometry, same flow conditions (DG-SF). Test geometry is *different* from the training dataset. Here, we test is CFDNet's capacity for generalization where the edges of the test geometries have not been seen previously by the CNN. To underscore that, the CNN models trained for the previous use case SG-SF are reused. The goal is to evaluate the performance of CFDNet on the NACA0012 airfoil [72], the geometry of which follows the mathematical definition in [28] and a cylinder shown in Figure 3.6.

The accuracy and performance of CFDNet are evaluated by comparing it against the traditional physics solver-only simulation. First, the evaluation of the accuracy of the CNN prediction and the end-to-end CFDNet coupled framework. Then, the evaluation of the performance of CFDNet and finally examine the importance of the *warmup* method in accelerating the convergence of CFDNet, especially for new test geometries.

3.3.1 Model and CFDNet accuracy

One of the goals of this work is to accelerate flow simulations using DL without relaxing the convergence constraints of the physics solver. Since the network is trained to predict the final steady-state (iteration N), it is hypothesized that the inference is "close" to the final solution. In this section, this hypothesis is tested by comparing the CNN model's output with CFDNet's output, that is, compare the solution with and without *refinement*. At the same time, we also compare both (CNN model and CFDNet) to the physics solver solution qualitatively and quantitatively.

Figure 3.7 shows the qualitative results for the OG-DF interpolated case. The leftmost column presents the velocity field after *warmup* +*inference* (that is, the CNN model's out-

put). We observe the similarity between the output from the CNN and the physics solver solution in the third column. However, this is not the case for the DG-SF airfoil at $Re = 6 \times 10^5$. It is seen that the CNN predicted pressure field in Figure 3.8 (leftmost column) qualitatively differs from the physics solver solution (rightmost column) mostly in the front part of the airfoil. Similiar to the airfoil, it can be visually concluded that the prediction of the modified eddy viscosity for the DG-SF cylinder in Figure 3.9 with the CNN (leftmost column) is qualitatively far from the physics solver (rightmost column). 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 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 the rightmost column of Figure 3.7 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 3.2. This suggests that the CNN (that is, warmup + inference) is a promising approach as a pure surrogate when the geometry is the same (OG-DF) but not otherwise.

Recall from the determination of the convergence criteria of CFDNet that the quantitative analysis through the above error metrics alone is insufficient. Moreover, Table 3.2 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 has to satisfy the conservation of mass Equation (2.1), that is, $\nabla \cdot U = 0$. OpenFOAM provides a tool to numerically compute the divergence of the velocity for a user-given flow field. This tool is applied to the CNN model's predicted flow field. Table 3.3 compares the result of the tool for the flow fields from CNN model-only (*warmup* + *infer-*

ence), 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 3.3 illustrates how CFDNet and the physics solver yield a divergence-free field (that is, both satisfy conservation of mass), whereas the CNN model's output is far from the expected tolerance. 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.

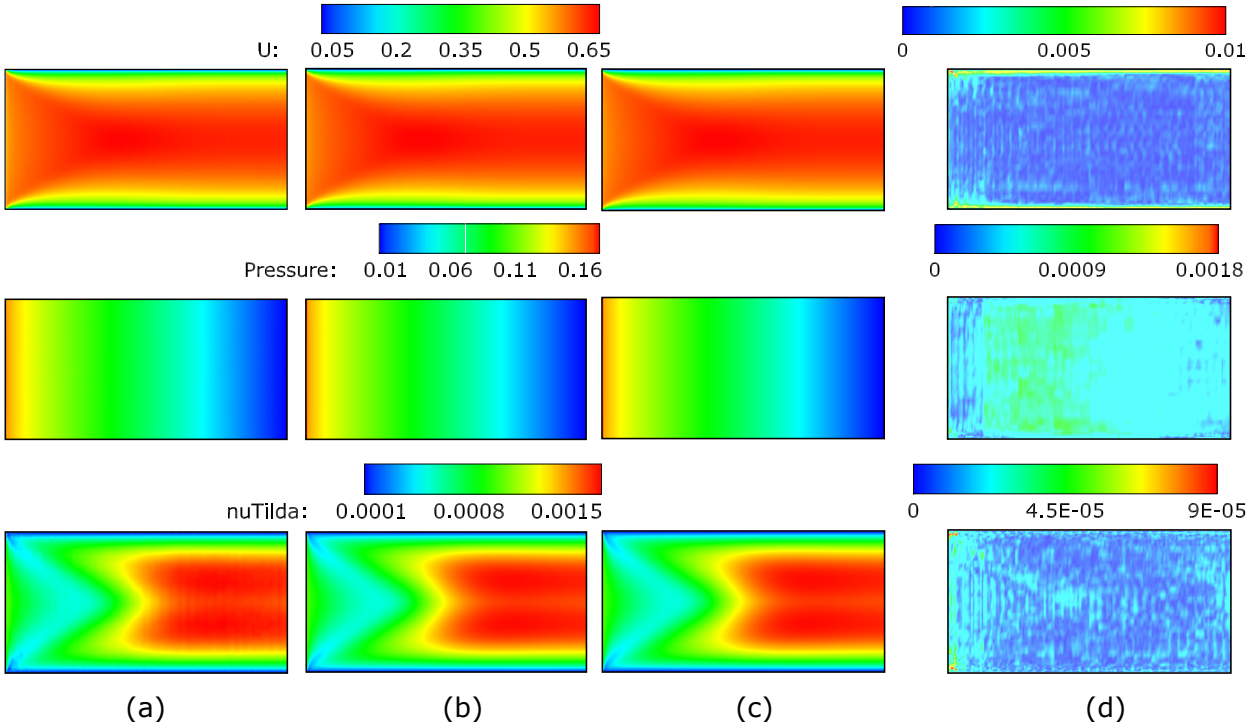


Figure 3.7: Velocity field in $\mathrm{m\,s^{-1}}$ for channel flow at $Re = 5600$ (up). Kinematic mean pressure in $\mathrm{m^2/s^2}$ (middle) and Modified eddy viscosity field in $\mathrm{m^2/s}$ (down). (a) *warmup* + *inference* (no *refinement*), (b) CFDNet, (c) physics solver in OpenFOAM, and (d) per-cell absolute error between (a) and (c).

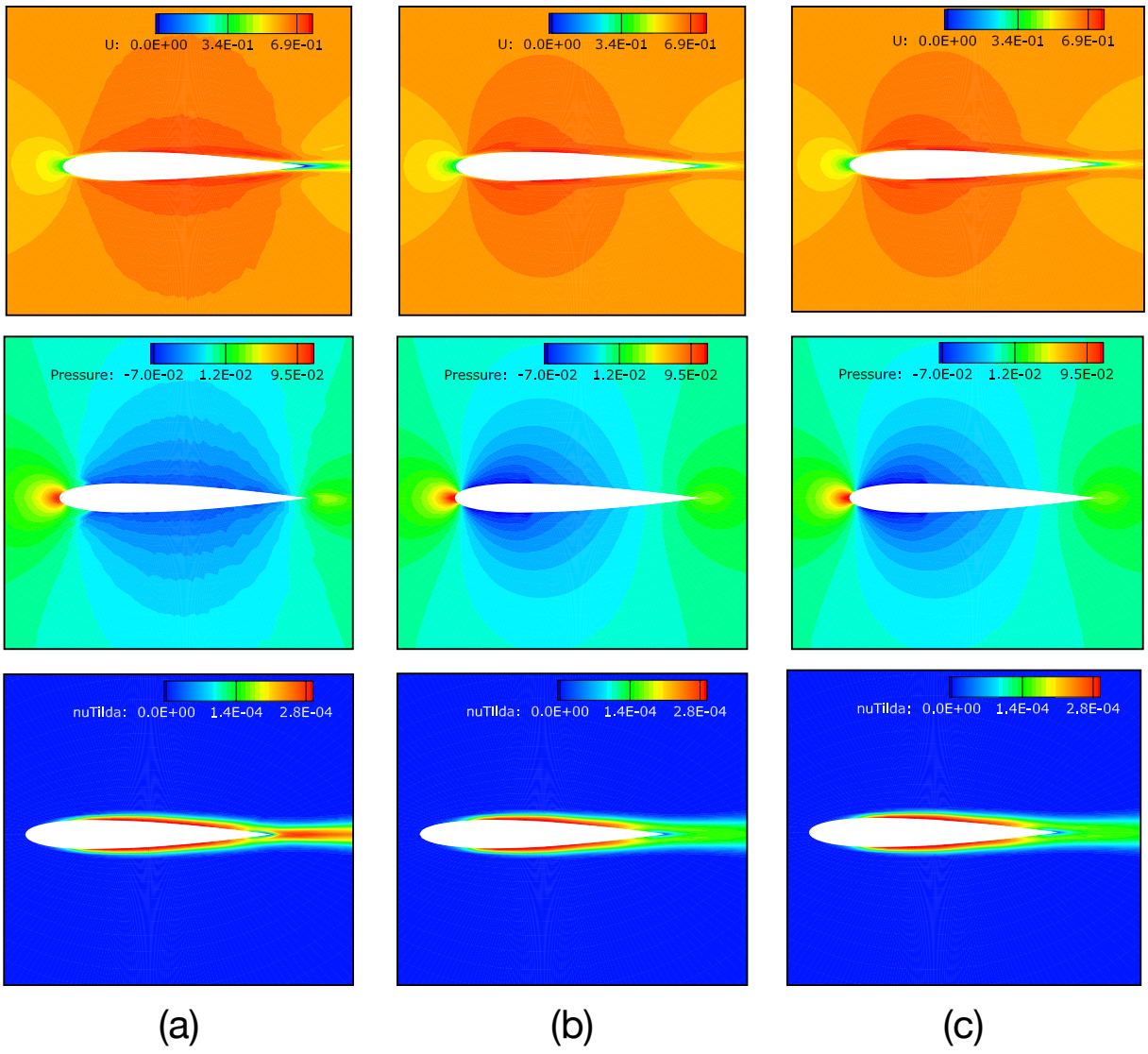


Figure 3.8: Velocity field in $\mathrm{m\,s^{-1}}$ around an airfoil at $Re = 6 \times 10^5$ (up). Kinematic mean pressure in $\mathrm{m^2/s^2}$ (middle) and Modified eddy viscosity field in $\mathrm{m^2/s}$ (down). (a) *warmup* + *inference* (no *refinement*), (b) CFDNet, and (c) physics solver in OpenFOAM.

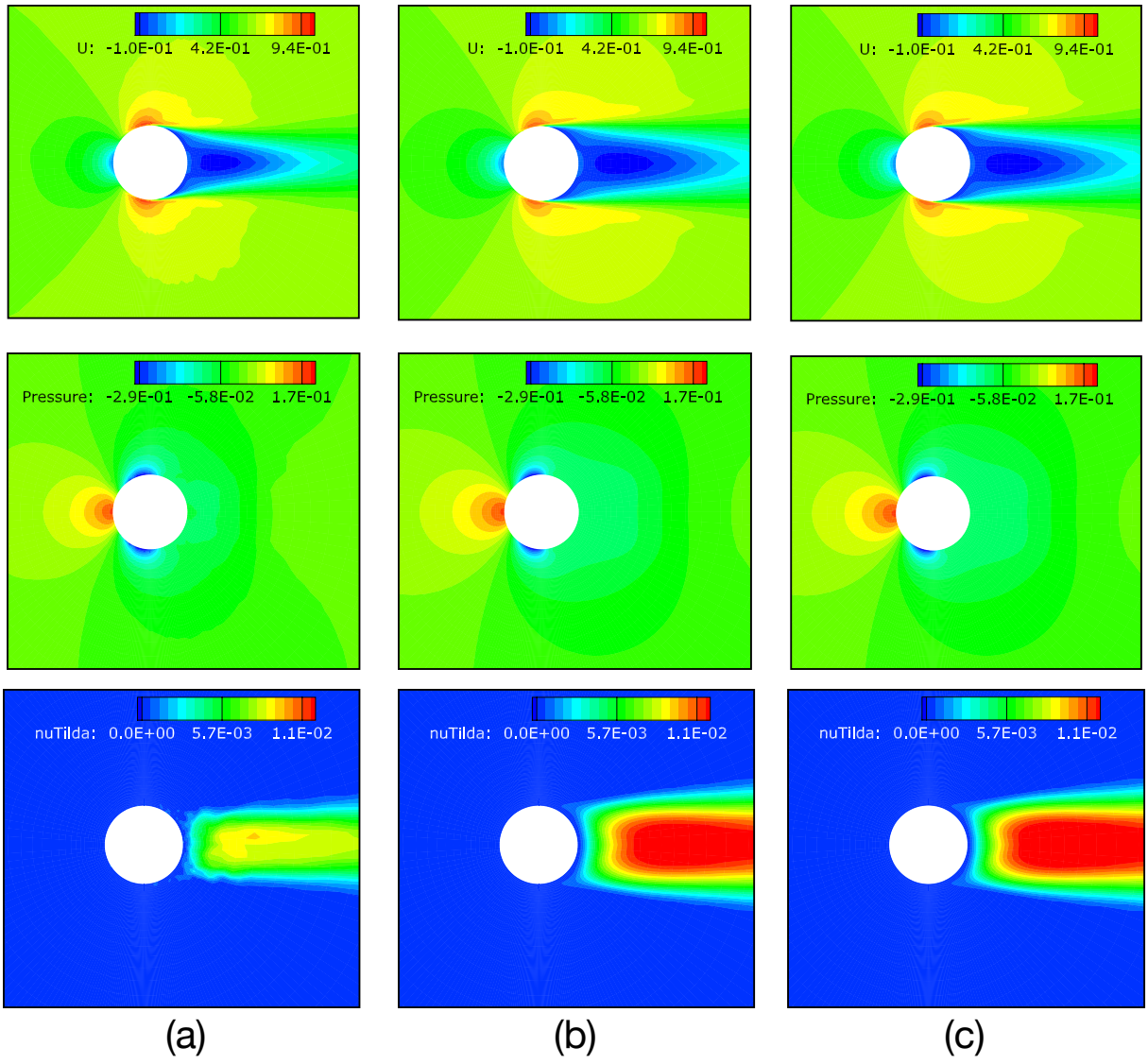


Figure 3.9: Velocity field in $\mathrm{m\,s^{-1}}$ around a cylinder at $Re = 6 \times 10^5$ (up). Kinematic mean pressure in $\mathrm{m^2/s^2}$ (middle) and Modified eddy viscosity field in $\mathrm{m^2/s}$ (down). (a) *warmup* + *inference* (no *refinement*), (b) CFDNet, and (c) physics solver in OpenFOAM.

Network	Metric	Best	QoI
Guo et al.[40]	RME	1.76%	$\bar{U}$
Thuerey et al.[42]	RME	<3%	$\bar{U}$, $\bar{p}$
Smart-fluidnet [54]	MAE	9×10^{-3}	ρ
TBNN [5]	RMSE	0.08	Reynolds Stresses
Tompson et al.[39]	L_2 norm	0.872	$\nabla \cdot U$
Maulik et al.[41]	L_1 norm	<2e-4	ν_t
CFDNet	RME	0%	$\bar{U}$, $\bar{p}$ and $\tilde{\nu}$

Table 3.2: Errors reported in the literature. Different works have considered different metrics - RME, mean absolute error (MAE or L_1 norm), root mean squared error (RMSE), L_2 norm. There is no consensus for an acceptable magnitude of error, and the errors reported as acceptable vary by work.

Test Case	*warmup*+*inference* only	CFDNet	physics solver
channel flow $Re = 5600$	3.12×10^{-4}	$< 1 \times 10^{-10}$	$< 1 \times 10^{-10}$
airfoil $Re = 6 \times 10^5$	8.15×10^{-3}	$< 1 \times 10^{-10}$	$< 1 \times 10^{-10}$
cylinder $Re = 6 \times 10^5$	9.8×10^{-3}	$< 1 \times 10^{-10}$	$< 1 \times 10^{-10}$

Table 3.3: Mass local error. The value reported is the average over all the cells. The ML model does not result in a divergence-free field. This is a major motivation for re-applying the physics solver after the initial warmup and inference steps. After the final refinement, the field once again is divergence-free.

3.3.2 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 (that is, K iterations). The physics solver is run in parallel as described in Section 3.2.

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 have been detailed in Section 3.2, (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 (that is, the output tensor after inference) four orders of magnitude for all variables in test sets B and C. For test set A, the residual is dropped 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 3.10 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 (that is, K for warmup, M for inference, and N for the physics-only solver). First, we 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 for each test case represents the time taken by CFDNet and the rightmost bar (colored gray) 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

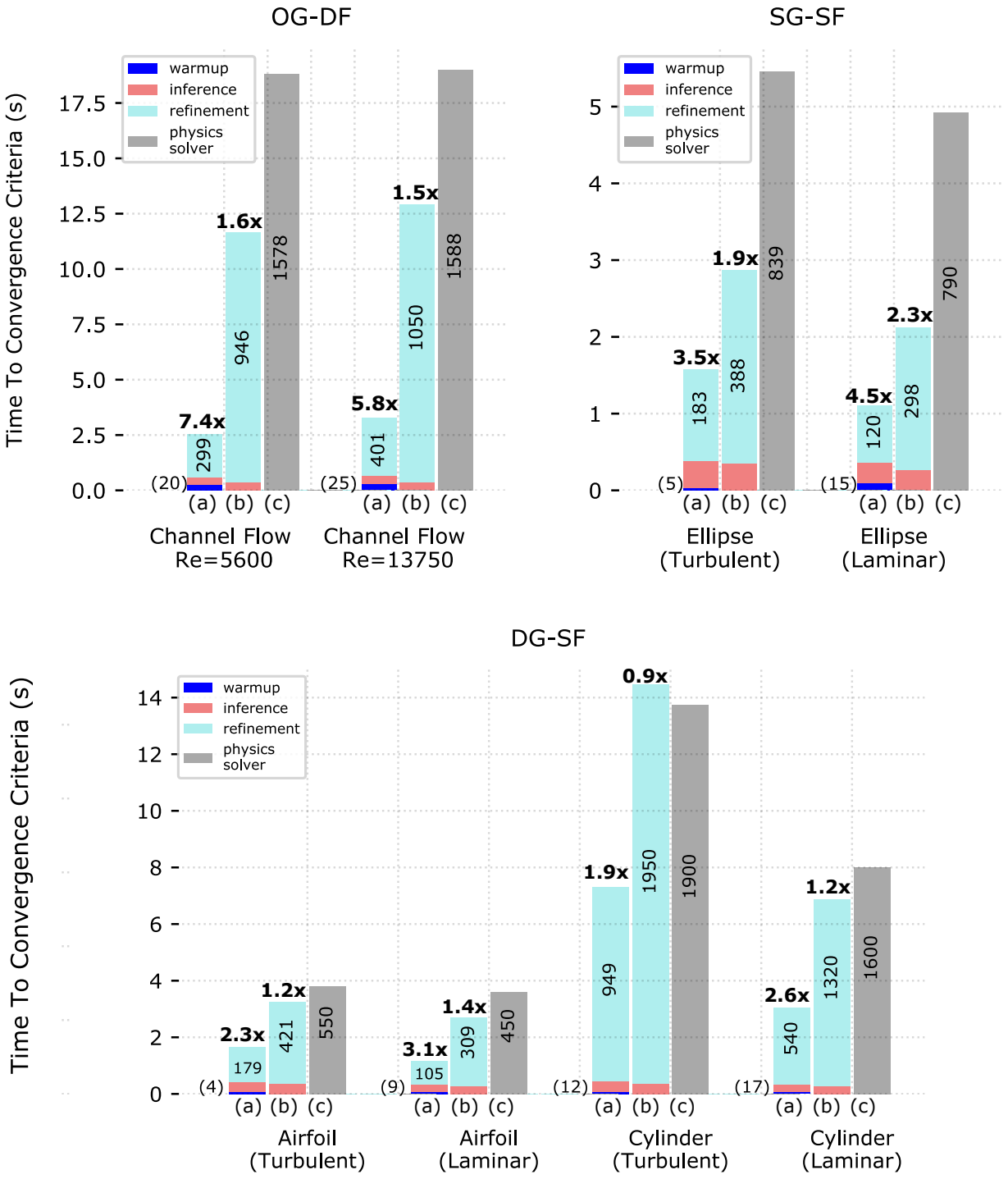


Figure 3.10: Breakdown of running time to convergence for each test case. Time to convergence of: (a) CFDNet w/ *warmup*, (b) CFDNet w/o *warmup*, and (c) the physics solver. The values inside the bars are the number of iterations it took for that stage. In parenthesis, the *warmup* number of iterations. At the top of each column, the speedup with respect the physics solver.

simulation which requires 1578 iterations (1588 for extrapolated flow conditions) to drop the residual to 1×10^{-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). It is observed 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 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 $1.9 - 2.6\times$ 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. Now, the focus is on comparing the results presented in the previous section when the inference is done without *domain-specific* knowledge (no *warmup*). The (b) bars in Figure 3.10 show the time taken by CFDNet when the warmup method is not applied. Without warmup, the speedups drop significantly to $1.2 - 2.3\times$.

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 (for example, 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 quantitatively[4] 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, even a slowdown is observed, compared to the physics solver without warmup. To summarize, across all the flow configurations tested in this work, warmup only takes $1-2\%$ of the overall iterations of the physics solver while yielding significant speedups of $1.9-4.6\times$.

3.4 Related work

Accelerating fluid simulations with surrogates. Shape design optimization relies on the minimization of an objective function. GP approximate this objective function to be computationally tractable. However, these models rely on a specific geometry parametrization [22] which limits its generalizability. Other regression techniques, such as MARS and polynomial regressions [23, 33, 34], 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. With CFDNet, we take an alternate approach with a CNN that predicts the velocity and the pressure fields for all points in the flow field.

Accelerating fluid simulations with NNs. In the past few years, several researchers have

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

leveraged NN to accelerate CFD simulations. In [39], the authors 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. Alternatively, Guo et al. [40] find a real-time solution to viscous laminar flows around solid objects. However, the above approaches have elemental constraints. First, Eulerian fluid simulations ignore second-order velocity derivatives in the NS 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 [1].

There have been recent attempts to find NN-based accelerators for turbulent flows. The results are promising but the NNs 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 [41]. Thuerey et al. use a novel input-output representation but their approach does not account for the eddy viscosity field [42]. 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 [42] 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 NN-based mapping back into the physical solver to enable both extrapolation and generalization with the same model without relaxing the convergence constraints.

3.5 Conclusions

We have shown that coupling RANS fluid flow simulations with a CNN, the CFDNet framework, can significantly accelerate the convergence of the overall scheme. CFDNet speeds the simulations up by a factor of 1.9× - 7.4× 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 has been 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.

First, a 64×256 grid size is a coarse resolution and needs to be scaled for real-world applications. We expect that the CNN trained in these case studies performs well for test cases that use the same grid size as the training set, and declines as we increase to higher resolutions. This is likely because higher resolutions present physical effects absent in meshes which the network has not learn. Hence, the current approach is not adequate for accelerating simulations in finer grids; the ones used in design optimization/exploration of real-world systems. A brute force solution to this limitation is to conduct data collection and training at higher resolutions. However, data collection times and training times at high resolutions are computationally intractable. This disadvantage needs to be circumvented to make the CFDNet approach a viable tool for most engineering systems of interest.

Second, CFDNet might present poor generalization capacity due to more complicated physics and complex geometries. The same CNN complexity (number of layers/filters/nodes) has been used to tackle two different case studies: (i) accelerate the simulation of an unseen flow

condition keeping the geometry constant (channel flow) and (ii) accelerate the simulation of an unseen solid body - keeping the flow conditions constant throughout all the experiments - in the external aerodynamics case. Clearly these two cases have very different complexities, the latter being a more challenging system. The reason is that even though the flow conditions are kept the same the flow regime accross the ellipses is very different. The results show how CFDNet works significantly better in the channel flow case. Results also show that the network is not able to distinguish if there is (cylinder) or there is not (airfoil) flow separation. Therefore, a significant amount of refinement iterations are required to correct these CNN-inferred flow fields. This indicates that the network needs more complexity to capture these differences and be more accurate in realistic design cases; for example, determining when an airfoil is at stall.

These limitations pose a challenge for the scalability and generalizability of this method. In the next chapter, we present SURFNet, a strategy to extend the scope of CFDNet to high-resolution flow.

Chapter 4

SURFNet: A transfer learning framework for super-resolution of fluid flows

To account for more aspects of the physical phenomena being modeled, practitioners necessitate an increase in the resolution of the system resulting in high computational costs. Figure 4.1 shows the time-to-solution with increasing grid size for a turbulent flow simulation around the NACA0012 airfoil. A dual-socket, 40-core system requires ten seconds at a resolution of 64×256, and 100 minutes at 2048×2048 for a single airfoil shape in one flow configuration (one test case). In aircraft design, there are typically many airfoil shapes [73], each of which requires simulations performed across a range of parameters such as Re , various angles of pitch, yaw, and roll to account for rotation as well as different angles of attack, resulting in thousands of test cases. To address this challenge, *super-resolution* is an approach to **reconstruct fine-scale flow physics from coarse-grid solutions.**

Most recent applications of super-resolution employ DL [50–53, 74–77]. Table 4.1 summarizes

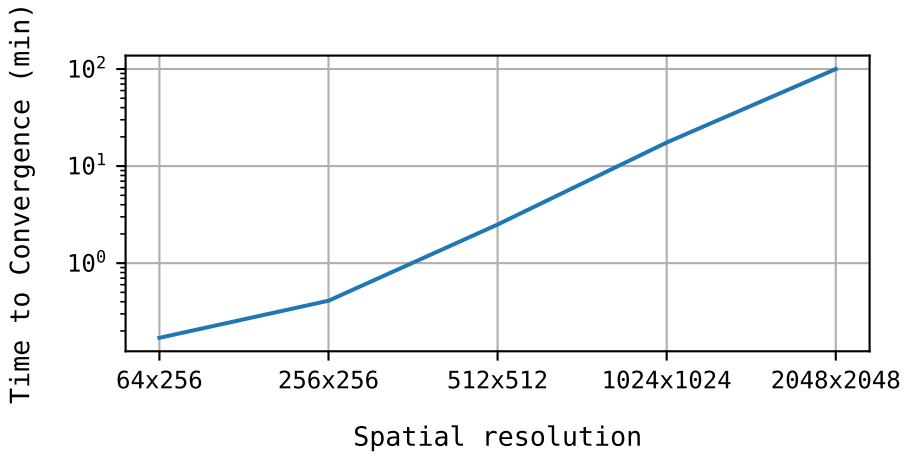


Figure 4.1: Time to convergence for flow around a NACA0012 airfoil at different spatial resolutions on a dual-socket Intel Xeon Gold 6148 CPU (total 40 cores)
.

the SOTA approaches for DL-based super-resolution and categorizes them across several features.

First, the majority of CNN-based approaches are finite-dimensional maps and hence lack resolution-invariance [40, 42, 64, 78, 79]. Mesh-free, resolution-invariant DL methods are a potential alternative to query fine-scale flow physics [50–52]. However, these approaches use low-resolution data downsampled from high-resolution solutions to train the network. As a result, current mesh-free methods suffer from the same limitation as classical CNN-based approaches that require a large number of computationally expensive simulations to collect training data at high resolutions. Practical CFD simulations require high spatial resolutions (such as 1024×1024) according to NASA's FUN3D and CFL3D solvers [70] making current DL approaches computationally prohibitive (see Figure 1). Second, most approaches [50–53, 77, 78] lack generalization to unseen geometries where the test geometry is a subset of the training data – a key limitation for CFD researchers/practitioners. Third, only a limited number of approaches support super-resolution of turbulent flows that are significantly more challenging than laminar flows [77] and ubiquitous with a wide range of applications in aerodynamics, atmospheric science, turbomachinery, and propulsion [42, 53, 64, 79]. Lastly, it's common to use DL algorithms as a pure surrogate model that

Related Work	Target Flow (PDE)	Test Geometry Unseen in Training	Meets convergence constraints	Resolution Invariant	Training data collected on coarse-grids	Highest Test Spatial Resolution	Error metric	Error value (best)	Method
Fourier Neural Operator [50]	Darcy	🔴	🔴	🟢	🔴	421×421	RE	1×10^{-2}	NO
Fourier Neural Operator [50]	Navier-Stokes	🔴	🔴	🟢	🔴	256×256	RE	8.6×10^{-3}	NO
Graph Kernel Network [51]	Darcy	🔴	🔴	🟢	🔴	241×241	L_2	3.7×10^{-2}	NO
MeshFree FlowNet [53]	Rayleigh-Bénard	🔴	🔴	🟢	🔴	4×512	NMAE	3.3×10^{-3}	CNN
Gao et al. [77]	Laminar	🔴	🔴	🟢	🔴	200×200	RE	0.025	CNN
TFNet [78]	Rayleigh-Bénard	🔴	🔴	🔴	🔴	1792×256	RMSE	200	CNN
Guo et al. [40]	Laminar	🟢	🔴	🔴	🟢	128×256	RME	1.76%	CNN
CFDNet [79]	Navier-Stokes	🟢	🟢	🔴	🟢	64×256	RME	0%	CNN
Smart-fluidnet [54]	Eulerian	🟢	🟡	🟢	🟡	1024×1024	MAE	9×10^{-3}	CNN
SURFNet	Navier-Stokes	🟢	🟢	🟢	🟢	2048×2048	RME	0%	TL

Table 4.1: Comparing SOTA approaches in DL for 2D CFD simulations on nine different features. SURFNet is a novel network-based transfer learning (TL) framework that (1) generates accurate solutions up to 2048×2048 spatial resolutions for turbulent flows from low-resolution models, (2) generalizes to unseen-in-training geometries, (3) meets the original convergence constraints of traditional CFD solvers, and (4) collects data at low-resolution on coarse grids for training – whereas prior works target non-turbulent or non-viscous flows [40, 51, 52, 54], only test on geometry domains that were part of the training phase [50, 53, 78], replace partly [54] (left yellow dot) or entirely [40, 50, 53, 78] traditional solvers with a neural network surrogate not meeting convergence constraints, and most importantly, train with data downsampled from high-resolution simulations or are unsupervised (right yellow dot) [50–54]. *NO* stands for Neural Operator and *CNN* for Convolutional Neural Network.

entirely replaces the CFD solver, thereby not providing the same convergence guarantees as the latter. There is a pressing need to design DL based models for super-resolution of turbulent flows that (a) eliminates the need for extensive data collection at high resolutions, (b) provides discretization and resolution-independent acceleration, and (c) can generalize to unseen geometries and flow configurations while meeting the convergence guarantees of the traditional physics solvers.

To address the above need, in this thesis we present **SURFNet** (**SU**per-**R**esolution **F**low **Net**work), a novel approach to reconstruct fine-scale flow physics from coarse grid data by primarily training the DL model on *low-resolution* inputs. Using this *coarse-model*, SURFNet *transfer learns* the model on high-resolution turbulent flow solutions, significantly reducing the overall data collection time and the total size of the training set. To enable efficient super-resolution, we propose two variations of transfer learning.

1. One-shot transfer learning (OSTL): learning is conducted from the coarse model to the target resolution in a single shot.
2. Incremental transfer learning (ITL): learning is done step-by-step incrementally on the training set of intermediate discretizations up to the target resolution.

To empirically evaluate SURFNet, we again solve the RANS equations (2.1), (2.2), (2.3).

4.1 Transfer learning for super-resolution of flow simulations

Our objective is to accelerate the convergence of high-resolution turbulent flow simulations. We achieve this by reconstructing fine-grid flow solutions from coarse grid models referred to as *super-resolution* with minimum data collection at higher resolutions.

In this section, we first describe the design of the CNN architecture to train the *coarse model* with low-resolution data. Then, we present SURFNet, a novel transfer learning-based super-resolution approach that relaxes the demand for generating expensive and time-consuming training data to train accurate models for discretizations at higher resolutions.

4.1.1 Coarse grid network

Recall that one of CFDNet's disadvantages was its limited generalization capacity to unseen-during-training geometries. Here, we tackle this limitation and augment the design of the previous CNN to generalize better to unseen test cases, including unseen geometries and unseen non-trivial flow conditions. We hence aim to learn a generalizable model that can predict flow around rotated geometries (that is, varying pitch angle, θ) and changing flow fields (that is, angle of attack, α) as they are critical features in design exploration and shape optimization. For instance, it is common practice to simulate the flow field by varying the angle of the incoming flow and rotating the airfoil geometry [80].

We augment CFDNet's CNN architecture and design an eight-layer, convolutional - deconvolutional neural network - 4 convolution layers followed by 4 deconvolution layers. The number of filters are, in order: 16, 32, 128, 256, and 256, 128, 32, 16. An illustration can be seen in Figure 4.2. We use a deeper network compared to CFDNet (8- versus 6-layer CNN) because the training dataset is larger and has more features to learn (for example, rotation). All layers have a filter size of 5×5 and a stride of 1. This overlap captures both the short and long (spatial) range dependencies between the flow variables while covering all regions of the flow field present in the input image. We use the LeakyReLU activation function for all layers because the output image contains real-valued variables.

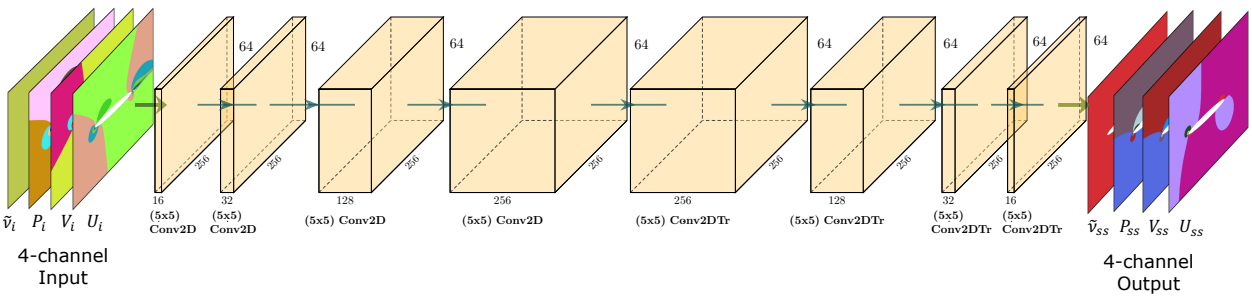


Figure 4.2: CNN and its input-output representation. The CNN is a symmetric, fully convolutional-deconvolutional neural network. The input and the output are , like in CFD-Net, a 4-channel tensor image: each channel represents one flow variable (x-velocity U, y-velocity V, pressure P and modified eddy viscosity $\tilde{\nu}$). The difference between the input and the output tensors is that the former has the flow values of an intermediate iteration (i), while the latter is the flow field at steady-state (ss).

4.1.2 Why transfer learning?

The CNN in Figure 4.2 relies on intermediate and final solutions of flow simulations to train the solution operator, G. The ideal scenario in machine learning is the availability of vast amounts of labeled training data for supervised models. However, both collection of large-scale high-resolution data and training with large input sizes are prohibitively expensive (see Figure 4.1). An alternate approach is to use the convolutional operator calibrated with low-resolution data to predict high-resolution solutions. Although this is feasible, the accuracy reduces dramatically, especially for turbulent flows as we show empirically in Section 4.3. Therefore, CNN-based approaches demand end-to-end re-training for different resolutions and discretizations to achieve constant error.

The above limitations motivate the use of *transfer learning* to both solve the problem of insufficient training data at higher resolutions and to achieve resolution-invariance across discretizations. Transfer learning is a popular technique that relies on transferring a trained model across different learning tasks or from one model to another [81, 82]. It has been successfully applied to different applications such as drug discovery [83], disease detection [84], NLP [85], and machine fault diagnosis [86].

We propose another novel application of transfer learning that leverages the correlations inherent between low- and high-resolution inputs of turbulent flows. Low-resolution solutions of fluid simulations contain critical information that can be effectively extracted for high-resolution flow prediction: low-resolution solutions capture all present flow structures [87], even complex ones (for instance, the wake region behind solid bodies after flow separation). However, they do not resolve them, as high-resolution grids do. Therefore, given a pre-trained CNN model on large datasets at low-resolution (that is, *coarse model*), our objective is to transfer the trained model to the target fine-grid discretization to append the unresolved flow information.

An advantage of our super-resolution transfer learning methodology is that the input is *truly* from coarse-grid simulations, not downsampled from high-resolution data as in the SOTA [50–52][1]. This eliminates the requirement for generating computationally demanding (and in some cases intractable) data to build a high-resolution model. In the rest of this section, we describe the transfer learning pipeline and how we account for fine-scale physics and dynamics.

4.1.3 Super-resolution with transfer learning

Given $r_c : x_c \times y_c$ where r_c is the coarse-grid resolution corresponding to the discretization $x_c \times y_c$, the super-resolution task of SURFNet is to recover solutions at fine-scales r_f : $x_f \times y_f \mid f = 1, ..., t$ where $x_f \times y_f$ is a fine-grid discretization and $r_f > r_c$, $\forall f$. We implement network-based transfer learning as illustrated in Figure 4.3 and propose two variations for super-resolution as described below.

One-Shot Transfer Learning (OSTL). In this approach, model weights are transferred

[1]Data downsampled from higher resolutions is not the same as data generated from coarse-grid solutions. Low-resolution data downsampled from high-resolution solutions contain information of resolved complex flow structures, whereas this information is inexistent in coarse-grid solutions.

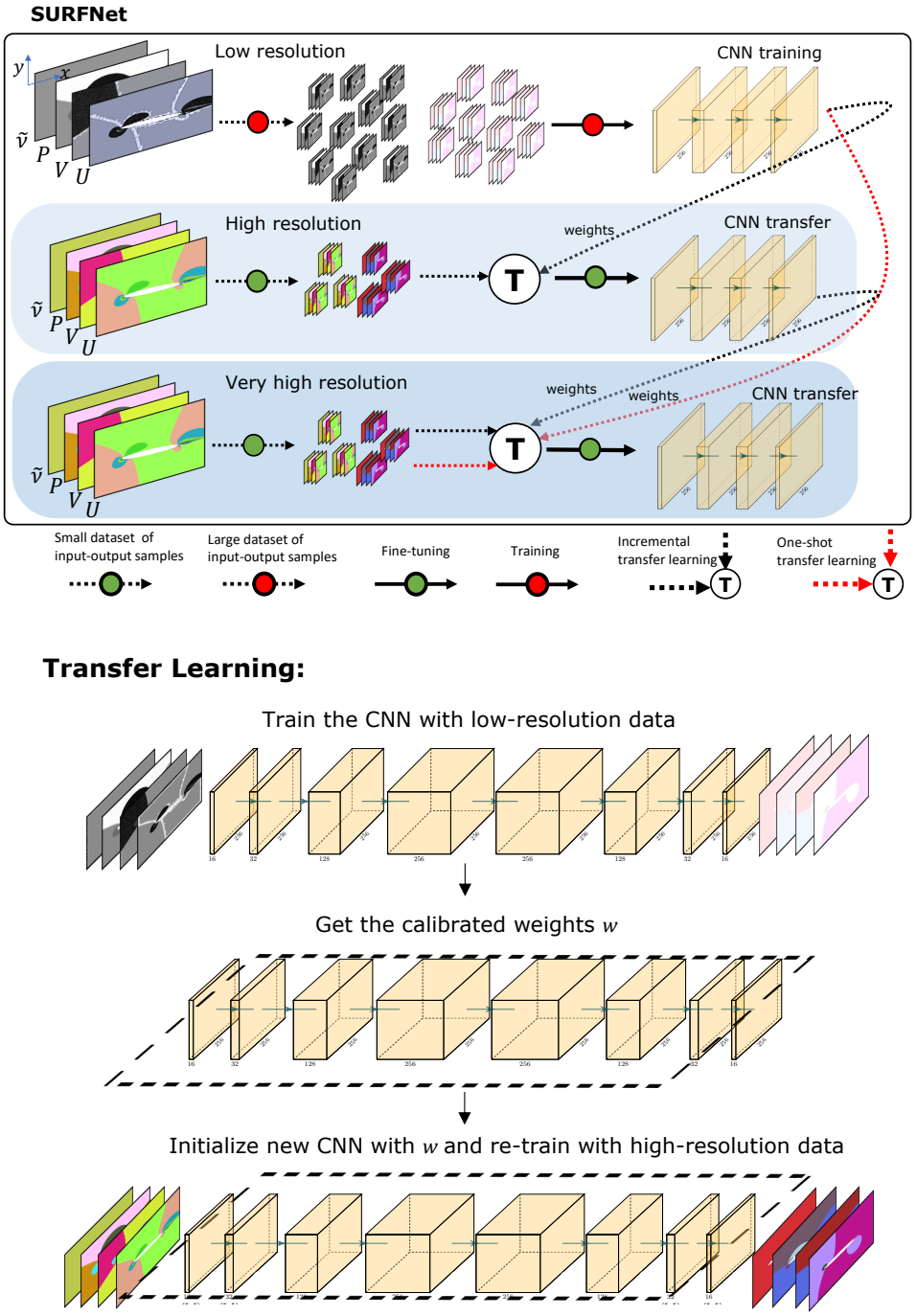


Figure 4.3: SURFNet (top) and its inductive transfer learning (bottom) for super-resolution of turbulent flows. A large dataset is collected at low resolutions to train the CNN and obtain the coarse model. Small data is collected at higher resolutions and SURFNet transfers the model weights from the coarse model to train the fine-scale model using either one-shot (red) or incremental transfer learning (black).

from the coarse model trained with r_c to the target resolution, r_t in a single training step. For example, if we desire to recover high-resolution flow fields at 2048×2048, we transfer learn from 64×256 to 2048×2048 in a *single shot*. This approach is illustrated in Figure 4.3 with a red, dashed line.

Incremental Transfer Learning (ITL). An alternative approach to OSTL is to train the CNN with the large-scale low-resolution dataset and perform transfer learning in a step-wise manner. That is, instead of transfer learning from r_c to the target resolution directly, r_t, we pass through intermediate resolutions step-by-step from the low resolution to the target high resolution (that is, r_c-> r_1-> r_2...-> r_t). For example, if we desire to recover high-resolution flow fields at 512×512, we transfer learn from 64×256 to 256×256, and finally from 256×256 to 512×512. A step size of 1 allows the model to incorporate new information from each intermediate discretization. We further discuss the impact of step size on the predictive accuracy in Section 4.3. The black dashed line in Figure 4.3 shows this variation of transfer learning. In ITL, the model learns from more data than OSTL without overfitting while still reducing the overall data collection since we avoid heavy data collection at any specific high resolution.

To recover the solution at the desired target discretization, we perform the steps outlined below for both transfer learning approaches (that is, OSTL and ITL).

1. *Coarse-grid data collection.* The training dataset is created by performing large-scale simulations at low-resolution discretizations, $x_c \times y_c$ using the physics solver detailed in Section 3.2.

2. *Training.* After generating low-resolution training data, the CNN in Figure 4.2 is trained with this large dataset. By carefully controlling for under- and over-fitting of the network with a validation dataset, we obtain the coarse model.

3. *Fine-grid data collection.* Due to the challenge of data acquisition at fine-scale discretiza-

tions, $x_f \times y_f$, we limit the number of simulations at high resolutions to the bare minimum to create the *transfer dataset*. In Section 4.3, we show that a single geometry is sufficient to transfer the coarse-grid features and recover the fine-scale physics.

4. *Transfer learning.* In our approach, we choose to implement inductive transfer learning, where we reuse the network (including its structure and parameters) that was pre-trained to learn the source model (that is, coarse model) [88]. Since the source and target domains are the same, we intuitively expect this technique to preserve the common features extracted between the two learning tasks. We treat the coarse model as a feature extractor of high-resolution flow fields. All layers of the CNN use a stride of 1 to not reduce the dimensionality of the input-to-output map during the low-resolution training phase. However, for high-resolution input-output pairs, this model *is* a low-dimensional representation – an extractor of the prevalent flow features across discretizations (for example, flow effects due to the boundary conditions, fields' shape in the free stream, and flow variations due to the change in α and θ). Since coarse discretizations do not have enough domain points to define accurate flow field solutions in areas of strong gradients, the features extracted need to be fine-tuned but not re-learned. The transferred network is updated by fine-tuning the weights as follows (see Figure 4.3).

a. Re-initialize the transferred network with the weights obtained from the coarse model for OSTL. For ITL, the transferred network is initialized with the weights from the largest model pre-trained at resolution r_f such that $r_f < r_t$.

b. Start training the transferred network with the transfer dataset. At this stage, it is critical to append the fine-grid flow features. Since we start with a good initial calibration of the weights from a pre-trained model of the same domain, only *fine-tuning* is required to update the weights of the transferred network. Fine-tuning of network parameters is done with a low learning rate (that is, small updates to the weights) and for very few epochs (1 or 2 at most) to avoid overfitting to the geometry (or geometries)

of the small transfer dataset while preserving the generalization capacity of the model.

In summary, SURFNet is a transfer learning-based super-resolution flow network that learns three distinct CNN models to reconstruct high-resolution turbulent flow fields – (1) the low-resolution coarse model, (2) the OSTL model, and (3) the ITL model.

4.2 Experiment setup

In this section, we first describe the case study to evaluate SURFNet's potential for super-resolution. Then, we describe the low- and high-resolution datasets for training, transfer learning, validation, and testing and outline the training process of the coarse model.

4.2.1 Case study

We consider the external aerodynamics cases that we introduced in Section 3.2. However, in real scenarios, the exploration involves different geometries (for instance, airfoil shapes) simulated under various flow configurations such as rotation of the solid body and wind angle. Therefore, to apply to a large class of CFD problems, challenges in generalizing to unseen geometries need to be addressed. We aim to evaluate SURFNet's ability to accelerate the flow around solid bodies such as airfoils and cylinder with different rotations. Accordingly, these geometries are excluded from the training and transfer learning datasets. We resolve turbulent flow around solid bodies at a Re of 6×10^5 using equations (2.1), (2.2), (2.3). This setup - the flow regime (turbulent flow), the geometries (extrapolating to NACA airfoils), and the grid resolutions - is representative of real NASA case studies [70].

4.2.2 Dataset creation

We create a total of 15 distinct datasets and perform the simulations of all flow configurations with the solver described in Section 3.2. Table 4.2 summarizes the datasets, including the composition, resolution, and scope of each dataset.

Datasets		Low resolution	Higher resolutions
Training	NoG	10	•
	NoFC	90	
	Type	ellipses	
Transfer	NoG	•	1
	NoFC		6
	Type		ellipse $AR = 0.1$
Validation	NoG	3	
	NoFC	9	
	Type	NACA0012, ellipse $AR = 0.22$, cylinder	
Test	NoG	4	
	NoFC	8	
	Type	NACA1412, NACA0015, ellipse $AR = 0.3$, cylinder	

Table 4.2: Summary of datasets. The training dataset is from low-resolution simulations. At all high resolutions we collect identical transfer, validation, and test datasets. NoG is for the number of different geometries, and NoFC is for the number of flow configurations (that is, total number of simulations).

Training dataset. First, we collect a large-scale, low-resolution dataset to train the coarse model, at a 64×256 resolution (a common resolution for low-resolution solutions of similar case studies [70]). The core of this dataset is formed by simulations of flow around 10 different ellipses obtained by changing the aspect ratio AR that is defined as the ratio of the vertical to the horizontal semiaxis length, as shown in Figure 4.4. The AR's considered for the training dataset are: 0.05, 0.07, 0.09, 0.1, 0.15, 0.2, 0.25, 0.35, 0.55, and 0.75. For each ellipse, we consider 9 flow variations: five different angles of attack, α, and four different pitch angles, θ, chosen randomly for each ellipse in the range between $-2 - 6^\circ$ for a total of 90 flow configurations. We choose from thin to thick ellipses and angles of attack to cover a

wide spectrum of physical phenomena that is commonplace in aerospace design exploration. For instance, the angle of attack is an important variable in determining the magnitude of the force of lift. The α angles are obtained by changing the direction of the flow while maintaining the angle between the chord of the solid body and the cartesian x-direction at 0°. The θ angles are obtained by pitching the nose of the solid body up or down and maintaining a flow direction at a 0-degree angle with its longitudinal axis. These configurations are illustrated in Figure 4.4.

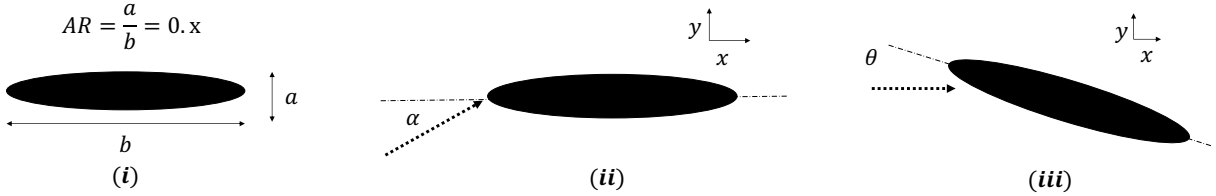


Figure 4.4: Geometry configurations used in training. The dotted arrow shows the direction of the incoming flow and the dashed line shows the chord of the solid body. Sketch of the (i) ellipse aspect ratio, (ii) angle of attack α, and (iii) pitch angle θ.

Validation dataset. We collect validation datasets to control the under- and over-fitting of the network during both the pre-training and fine-tuning phases. The validation dataset consists of flow around three different unseen-in-training geometries: an ellipse $AR = 0.22$, a symmetric NACA0012 airfoil, and a cylinder. For each geometry, we consider 3 flow variations – two α angles and one θ angle, chosen randomly as in the training dataset for a total of nine flow configurations. Note that we collect the same validation dataset at all resolutions, from 64×256 to 2048×2048.

Test dataset. We collect a new dataset to evaluate SURFNet's performance in accelerating high-resolution turbulent flow simulations. The test datasets consist of flow around four geometries: a non-symmetric NACA1412 airfoil, a symmetric NACA0015 airfoil, an ellipse $AR = 0.3$, and a cylinder. The airfoil and cylinder geometries are shown in Figure 4.5.

The test dataset contains geometries *unseen* during the pre-training or transfer learning phases. Nonetheless, some geometries are, a priori, more challenging than others. For

example, the non-symmetric NACA1412 airfoil has three unique features distinct from the training dataset: the flat trailing edge, the non-symmetry, and the chamber thickness (12% of the chord of the airfoil). Comparing the airfoil to the ellipse in the test set, the only feature unseen during the training phase is its chamber thickness ($AR = 0.3$). Moreover, even though a cylinder is a special case of an ellipse from a geometrical perspective, the physics in the rear of the cylinder has a large recirculation area not present in the training flows. For each geometry, we consider 2 flow variants: one α angle and one θ angle. Table 4.3 summarizes the different test cases. The test dataset is collected at all resolutions.

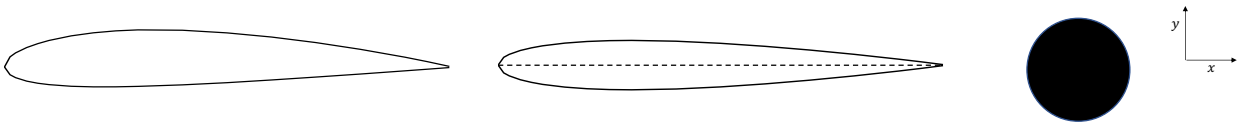


Figure 4.5: Non-symmetric NACA1412 airfoil (left), symmetric NACA0015 airfoil (center), and cylinder (right) as test geometries. The last two digits in the 4-digit NACA denomination represent the maximum thickness percentage of the chamber of the airfoil with respect to the airfoil's chord (dashed line).

4.2.3 Coarse model training

We train the CNN in Figure 4.2[2] using the training dataset described in Section 4.2.2 using double precision. We implement the CNN using Keras [71] and perform distributed training on four Tesla V100 GPUs connected with PCIe, using the TensorFlow 2.4 backend. No specific initialization is used in training. The batch size is 64, the optimizer is Adam, and the loss function is MSE. The learning rate is set to 1×10^{-4} with no decay for all training. The training is stopped using the EarlyStopping Keras callback by monitoring the validation loss with patience of 6 epochs. After 41 epochs, the training loss reaches 7×10^{-4} and validation loss reaches a value of 8×10^{-4}.

[2]We also considered a 10- and 12-layer CNN. However, adding additional layers did not improve performance.

4.3 Results and discussion

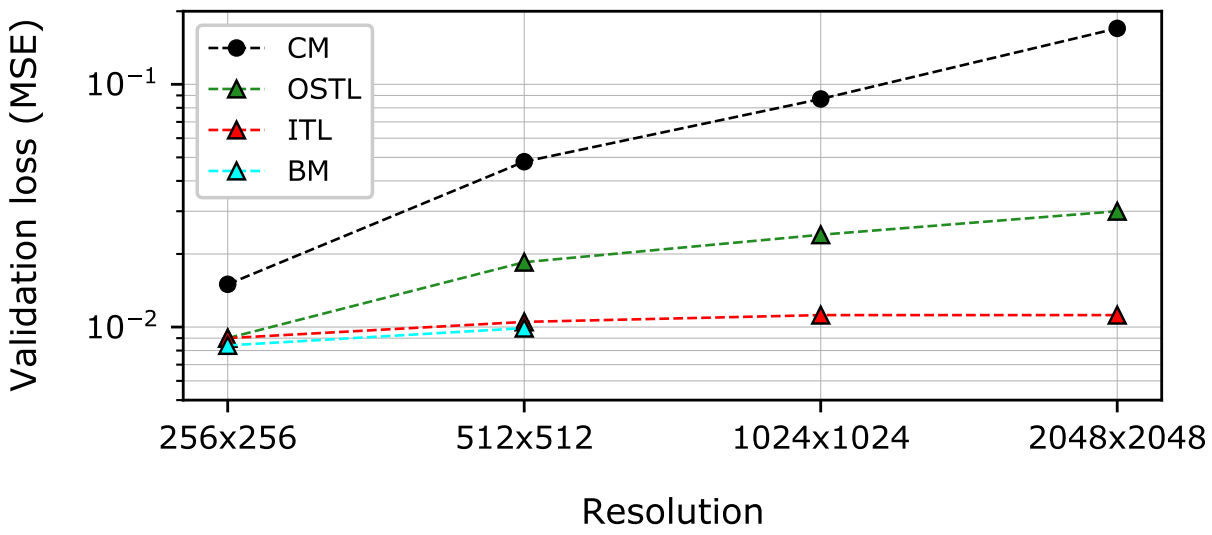
After pre-training and validating the coarse model, we perform fine-tuning using both transfer learning approaches (that is, OSTL and ITL) and evaluate SURFNet's ability for super-resolution at various target high-resolution discretizations. We start by empirically demonstrating the inefficiency of coarse model without fine-tuning to recover high-resolution turbulent flows, especially at fine-scale discretizations. Then, we evaluate SURNet's transfer learning and its ability to generalize to geometries unseen during the pre-training and fine-tuning phases. Finally, we evaluate its performance in reconstructing high-resolution turbulent flows with respect to the OpenFOAM physics solver. Besides, we also compare SURFNet against the baseline model (aka oracle), which performs full training with a large training dataset collected at higher resolutions.

4.3.1 Validation loss

One of our objectives is to maintain prediction accuracy across discretizations to build a resolution-invariant DL algorithm. Figure 4.6 shows the validation loss of the different models with increasing resolution size.

Coarse model loss. We observe that the validation loss of coarse model increases significantly with increasing resolution size from 1.5×10^{-3} at 256×256 to 2.1×10^{-1} at 2048×2048. These results indicate that coarse model trained with low-resolution data is unable to recover high-resolution flow fields. This lack of fidelity is because coarse discretizations learned with the coarse model are incapable of capturing sharp gradients and resolving flow instabilities prevalent at fine discretizations. Hence, the coarse model alone is inadequate for super-resolution.

OSTL loss. To augment the prediction capabilities of the coarse model, we perform one-



l

Figure 4.6: Coarse model (CM), one-shot transfer learning (OSTL), incremental transfer learning (ITL), and baseline model (BM) losses on the validation dataset at every target resolution.

shot transfer learning from the coarse model to the target high-resolution using the transfer dataset as discussed in Section 4.1. We set the learning rate to one order of magnitude lower than the training learning rate for the coarse model (from 1×10^{-4} to 1×10^{-5}), and train for *only* one or two epochs. The loss of the fine-tuned OSTL model on the validation dataset significantly reduces at all target resolutions compared to the coarse model as seen from Figure 4.6. At 256×256, OSTL reduces the validation loss from 1.5×10^{-2} for the coarse model to 9×10^{-3}, resulting in 60% overall reduction. We observe a similar trend with subtle improvements at larger resolutions. At 2048×2048 the validation loss reduces from 2×10^{-1} for the coarse model to 3×10^{-2} for OSTL – almost one order of magnitude.

We make two main observations from the OSTL results. First, the gap in the validation losses between OSTL and the coarse model increases with the grid size. Coarse discretizations do not contribute sufficient points in the domain to obtain accurate solutions. In areas of strong gradients, fine discretizations shape the flow field very differently from coarse discretizations as seen from Figures 4.7 and 4.8 for the NACA airfoils. The finer the discretization, the more distinct the resulting flow field is from the baseline low resolutions. Therefore, when

we transfer from the coarse model to very high resolutions (for example, 2048×2048), the new flow field seen during the transfer phase produces sizable changes to the weights of the network compared to transferring to mid-resolutions (for example, 256×256). Second, this divergence of the flow features between low- and high-resolution discretizations also explains why the OSTL validation loss increases with the grid size. Although OSTL substantially improves the accuracy of super-resolution, it still does not achieve resolution-invariance.

To alleviate these limitations, we consider three alternatives. First, to train for more epochs during fine-tuning. However, this comes with the risk of overfitting to the unique geometry in the transfer dataset, which is undesirable. Second, to include more geometries in the transfer dataset. This approach is also detrimental because it would require substantial data collection at high resolutions. Third, use incremental transfer learning (ITL). This approach is promising as the model is further fine-tuned with data at intermediate resolutions until the target discretization. We choose a step size of 1 to avoid the drawbacks of OSTL described above and improve the accuracy of the network at very high resolutions while still reducing the overall data collection.

ITL loss. SURFNet does incremental transfer learning using the same approach as in OSTL. Figure 4.6 plots the error of SURFNet after ITL on the validation dataset at each target resolution. SURFNet's OSTL and ITL approaches produce different validation loss values at every target resolution except 256×256. ITL improves the generalizability of the network. The validation loss drops by half at 512×512 and $3\times$ at 2048×2048 compared to OSTL. ITL reaches a loss that is invariant to the resolution: the validation loss remains constant at around 1×10^{-2} for every target discretization. Because the transfer dataset at each resolution is $15\times$ smaller than the training dataset, we still learn incrementally using far less data. Most importantly, ITL achieves similar accuracy to the baseline model or oracle trained using a dataset as large as the pre-training dataset for the coarse model at 256×256 and 512×512. This result is particularly notable because ITL reaches oracle-

level accuracy without the need for exhaustive training with large input sizes and large-scale high-resolution datasets. We present a more detailed performance comparison against the oracle in Section 4.3.3.

4.3.2 Performance analysis

We now study SURFNet's performance in super-resolution of turbulent flows. Recall that SURFNet's pre-training consists of training the base network with a large number of inputs of low-resolution data. The transfer phase adds a minimum amount of high-resolution data for fine-tuning the network. Therefore, in addition to performance, we also evaluate the ability of SURFNet in recovering the *same* high-resolution turbulent flow solution as the physics solver.

We test SURFNet in simulations of turbulent flow around 4 unseen-in-training geometries at 2 flow configurations each, for a total of 8 test cases at all target resolutions. Table 4.3 presents the comparison against the OpenFOAM physics solver. SURFNet creates three models – (i) low-resolution coarse model, (ii) OSTL model, and (iii) ITL model. Therefore, at every target resolution, we evaluate the time-to-convergence (TTC) using each model, namely: C-SURFNet, O-SURFNet, and I-SURFNet. We compare the TTC of each one of these models with the TTC of the physics solver (PS) and baseline model (BM)[3].

The TTC of the PS is computed by using in tandem two popular convergence criteria in the CFD literature [5]: (1) the residual of each flow variable drops 4 to 6 orders of magnitude and (2) by monitoring when physical quantities reach steady-state. In contrast, SURFNet reaches convergence in three stages. First *warmup* (W), where we start the simulation with the PS and let the residual drop between one and two orders of magnitude for each variable. This is sufficient for the fluid parameters close to the physical boundaries to capture the geometry

[3]A comparison is presented against the BM (or oracle) for only 256×256 and 512×512 due to the computational cost of collecting large datasets and training at higher resolutions.

		64 × 256		256 × 256				512 × 512					1024 × 1024				2048 × 2048			
Test case		PS	C-SN	PS	C-SN	O-SN	BM	PS	C-SN	O-SN	I-SN	BM	PS	C-SN	O-SN	I-SN	PS	C-SN	O-SN	I-SN
NACA 1412 $\theta = 5°$	TTC	0.16	0.08	0.4	0.22	0.2	0.2	2.4	1.7	1.3	1.2	1.2	17.2	14.3	10.1	8.6	95	95	59.4	47.5
	ITC	1865	825	3636	1874	1672	1669	7273	5018	3864	3460	3473	10118	8259	5779	4887	12338	12183	7556	6014
	S	1×	**2.1**×	1×	**1.8**×	**2**×	**2**×	1×	**1.4**×	1.8×	**2**×	**2**×	1×	**1.2**×	1.7×	**2**×	1×	**1**×	1.6×	**2**×
NACA 1412 $\alpha = 6°$	TTC	0.19	0.09	0.53	0.29	0.27	0.25	3.3	2.5	1.9	1.7	1.7	19.3	16.1	12.06	9.7	98.5	89.5	61.6	49.3
	ITC	2215	991	4818	2531	2263	2199	10000	7516	5706	4823	4830	11353	9289	6506	5504	12792	11474	7840	6241
	S	1×	**2.1**×	1×	**1.8**×	**2**×	**2.1**×	1×	**1.3**×	1.7×	**2**×	**2**×	1×	**1.2**×	1.6×	**2**×	1×	**1.1**×	1.5×	**2**×
NACA 0015 $\theta = 3°$	TTC	0.15	0.07	0.34	0.18	0.16	0.17	2.5	1.7	1.4	1.2	1.2	15.8	12.2	8.8	7.9	88.7	88.7	55.4	44.4
	ITC	1748	769	3091	1481	1326	1349	7576	4874	4032	3431	3424	9294	6977	4991	4475	11519	11365	7045	5605
	S	1×	**2.1**×	1×	**1.9**×	**2.1**×	**2**×	1×	**1.5**×	1.8×	**2.1**×	**2.1**×	1×	**1.3**×	1.7×	**2**×	1×	**1**×	1.6×	**2**×
NACA 0015 $\alpha = 0°$	TTC	0.13	0.06	0.32	0.18	0.16	0.16	2.3	1.6	1.3	1.2	1.2	14.5	10.4	8.5	7.3	82.4	82.4	51.5	41.2
	ITC	1515	626	2909	1470	1308	1311	6970	4802	3696	3308	3300	8529	5920	4845	4093	10701	10546	6533	5196
	S	1×	**2.2**×	1×	**1.8**×	**2**×	**2**×	1×	**1.4**×	1.8×	**2**×	**2**×	1×	**1.2**×	1.7×	**2**×	1×	**1**×	1.6×	**2**×
Ellipse AR=0.3 $\theta = 5°$	TTC	0.22	0.1	0.61	0.36	0.31	0.3	4.1	3.2	2.4	2.1	2.1	20.8	17.3	13.0	10.4	107.2	97.5	67.0	53.6
	ITC	2564	1158	5545	3116	2627	2615	12424	9381	7132	6036	6039	12235	10024	7475	5946	13922	12502	8546	6806
	S	1×	**2.1**×	1×	**1.8**×	**2**×	**2**×	1×	**1.3**×	1.7×	**2**×	**2**×	1×	**1.2**×	1.6×	**2**×	1×	**1**×	1.5×	**2**×
Ellipse AR=0.3 $\alpha = 1°$	TTC	0.25	0.13	0.64	0.36	0.32	0.33	4.6	3.3	2.6	2.3	2.3	22.3	17.2	13.1	11.2	114.7	104.3	71.7	57.4
	ITC	2914	1394	5818	3086	2763	2777	13939	9780	7568	6793	6790	13118	9918	7544	6387	14896	13387	9155	7293
	S	1×	**2**×	1×	**1.8**×	**2**×	**2**×	1×	**1.4**×	1.8×	**2**×	**2**×	1×	**1.3**×	1.7×	**2**×	1×	**1.1**×	1.6×	**2**×
Cylinder $\theta = 0°$	TTC	0.30	0.19	0.72	0.42	0.36	0.36	5.2	3.7	2.9	2.6	2.5	30.6	25.5	18.0	15.3	165.4	165.4	103.4	82.7
	ITC	4663	2157	6545	3704	3127	3127	15758	11079	8578	7702	7691	18000	14828	10416	8828	21481	21326	13270	10585
	S	1×	**2.1**×	1×	**1.7**×	**2**×	**2**×	1×	**1.4**×	1.8×	**2**×	**2.1**×	1×	**1.2**×	1.7×	**2**×	1×	**1**×	1.6×	**2**×
Cylinder $\alpha = 1°$	TTC	0.36	0.16	0.68	0.40	0.34	0.34	5.0	3.6	2.9	2.5	2.5	27.1	22.6	16.9	13.6	153.0	139.1	95.6	76.5
	ITC	4196	1844	6182	3490	2945	2941	15152	10646	8736	7399	7400	15941	13112	9791	7799	19870	17909	12264	9780
	S	1×	**2.2**×	1×	**1.7**×	**2**×	**2**×	1×	**1.4**×	1.7×	**2**×	**2**×	1×	**1.2**×	1.6×	**2**×	1×	**1.1**×	1.6×	**2**×

Table 4.3: Summary of the performance results. TTC is the time-to-convergence and ITC is the number of iterations-to-convergence of the physics solver (PS), C-SURFNet (C-SN), O-SURFNet (O-SN), I-SURFNet (I-SN) and the Baseline Model (BM). The speedup of all SURFNet models is calculated with respect to the physics solver.

		64× 256		256× 256		512× 512		1024× 1024		2048× 2048	
Test case		W	I	W	I	W	I	W	I	W	I
NACA 1412 $\theta = 5°$	T	2e-3	3e-3	3e-3	1.3e-2	1.5e-2	0.05	0.11	0.25	0.77	1
	NI	25		26		45		65		100	
NACA 1412 $\alpha = 6°$	T	2e-3	3e-3	3e-3	1.3e-2	1.6e-2	0.05	0.13	0.25	0.85	1
	NI	26		26		49		78		110	
NACA 0015 $\theta = 3°$	T	2e-3	3e-3	3e-3	1.3e-2	1.7e-2	0.05	0.122	0.25	0.7	1
	NI	25		27		50		72		91	
NACA 0015 $\alpha = 0°$	T	3e-3	3e-3	4e-3	1.3e-2	2e-2	0.05	0.14	0.25	0.92	1
	NI	31		39		60		80		119	
Ellipse $\theta = 5°$	T	3e-3	3e-3	4e-3	1.3e-2	2e-2	0.05	0.14	0.25	0.85	1
	NI	35		38		69		80		110	
Ellipse $\alpha = 1°$	T	2e-3	3e-3	4e-3	1.3e-2	1.8e-2	0.05	0.12	0.25	0.76	1
	NI	28		32		55		73		99	
Cylinder $\theta = 0°$	T	3e-3	3e-3	6e-3	1.3e-2	2.5e-2	0.05	0.14	0.25	1.05	1
	NI	40		50		75		84		140	
Cylinder $\alpha = 1°$	T	3e-3	3e-3	6e-3	1.3e-2	2.5e-2	0.05	0.14	0.25	1.05	1
	NI	42		50		76		83		142	

Table 4.4: Warmup (W) and inference (I) times (T) and number of iterations (NI) for each test case at each spatial resolution. Times are reported in minutes.

of the new problem. Next *inference* (I), where we use these intermediate flow variables as input to the CNN, which infers the steady-state. Finally, the CNN's output is constrained with the PS in *refinement* to reach the same convergence criteria as the PS [79]. The TTC of SURFNet is the sum of the time spent in the three stages. The three stages are run in parallel on the CPU described in Section 3.2 for a fair comparison against OpenFOAM's solver (which doesn't support GPU acceleration). Table 4.4 presents the time spent in W and I and the number of iterations in W.

C-SURFNet versus physics solver. The first column in Table 4.3 presents the eight cases in our test dataset. The results of C-SURFNet at 64×256 indicate good generalization capacity at the lowest resolution to *unseen* geometries. We observe consistent speedups around $2 - 2.1\times$, independent of the geometry or flow configuration. On the other hand, the first row shows the results for the first test case, flow around a NACA1412 at $\theta = 5°$ at different spatial resolutions. Although we observe significant speedups at low resolutions and C-SURFNet exhibits improved TTC compared to the PS, its performance gain also degrades

consistently with increasing discretizations. The speedup drops from 2.1× at 64×256 to 1× at 2048×2048, where it's no better than the PS. We observe a similar trend across all the test cases. These results are in accordance with the observations in Section 4.3.1, where the coarse model that has learned from only low-resolution inputs is incapable of predicting accurate high-resolution solutions.

O-SURFNet versus physics solver. In Table 4.3, all high resolutions have an O-SURFNet (O-SN) column, and we make three main observations from the results. First, O-SURFNet outperforms C-SURFNet for all test cases and resolutions. For instance, for the symmetric NACA0015 at $\theta = 3°$ at 2048×2048, the TTC of the PS is 88.7 minutes. O-SURFNet reduces this time to 55.4 minutes resulting in a 1.6× speedup (C-SURFNet achieved no performance gain). Figure 4.7 shows the corresponding qualitative results of O-SURFNet in the task of super-resolution. It resolves the fields of all fluid variables - velocity, pressure, and the modified eddy viscosity - at 2048×2048 faster than the PS while being pre-trained with low-resolution data with *only* fine-tuning at the highest spatial resolution. Second, O-SURFNet's performance is better at lower resolutions than higher resolutions. For the same NACA0015 test case, at 1024×1024, 512×512, and 256×256, the performance gains are 1.7×, 1.8×, and 2.1× respectively. We observe a similar trend of decaying performance with increasing resolutions across all test cases similar to the coarse model, albeit not to the same extent. This is in line with the results presented in Section 4.3.1 where OSTL losses increase the more dissimilar the target flow field is from the low-resolution flow. OSTL at 256×256 discretization achieves a 2× speedup irrespective of the test case, demonstrating its potential to generalize to higher resolutions provided the target flow features have sufficient overlap with the tiny resolution used for pre-training the coarse model from which it fine-tunes. However, this is also an indication that OSTL is incapable of achieving resolution-invariance. Third, the speedups achieved by O-SURFNet remain constant and stable among test cases (for instance, we observe speedups around 1.8× at 512×512), indicating that fine-tuning the model did not result in overfitting to the transfer geometry.

Neither the coarse model nor OSTL yield overfitted models and exhibit stable generalization capacities.

I-SURFNet versus physics solver. O-SURFNet's accuracy decreases as the resolution increases. An ideal solution is one that maintains accuracy (that is, resolution-invariant) while simultaneously requiring limited computational resources. In Table 4.3, we observe that across the board (that is, all test geometries and flow configurations unseen during pre-training and transfer learning), I-SURFNet achieves a $2 - 2.1\times$ speedup against the PS. Not only does I-SURFNet maintain the performance gain over PS across the different test cases but also across all target resolutions demonstrating both *generalization* and *resolution-invariance*. Since ITL incrementally fine-tunes the model, it requires more data than OSTL. However, fine-tuning is done on the same geometry across resolutions with $15\times$ lesser data, thereby significantly reducing the overall data collection at high spatial discretizations (compared to prior approaches that require considerably more high-resolution data). Figure 4.8 shows the qualitative results of I-SURFNet's flow solution around the nose of the non-symmetric NACA1412 airfoil at $\theta = 5°$. I-SURFNet successfully recovers high-resolution turbulent flow simulations on a geometry with at least three distinct features (that is, flat trailing edge, non-symmetry, and different chamber thickness) not present in the training or transfer datasets. This validates that SURFNet pre-trained with low-resolution data with only fine-tuning can generalize to unseen geometries. The largest target resolution studied (that is, 2048×2048) is $\mathbf{256}\times$ the size of the tiny discretization (that is, 64×256) used in pre-training the coarse model to stress SURFNet's ability in super-resolution. It is guaranteed to converge to a unique solution as long as the problem is well-posed [89].

We additionally explored the effect of the required size of the spatial step size to maintain accuracy for ITL. By incrementally fine-tuning with a step size of 1 up to 2048×2048, the performance gain is consistently $2\times$. This gain is the best possible achievable as observed by the results of the oracle or BM that is fully trained at 256×256 and 512×512 resolutions.

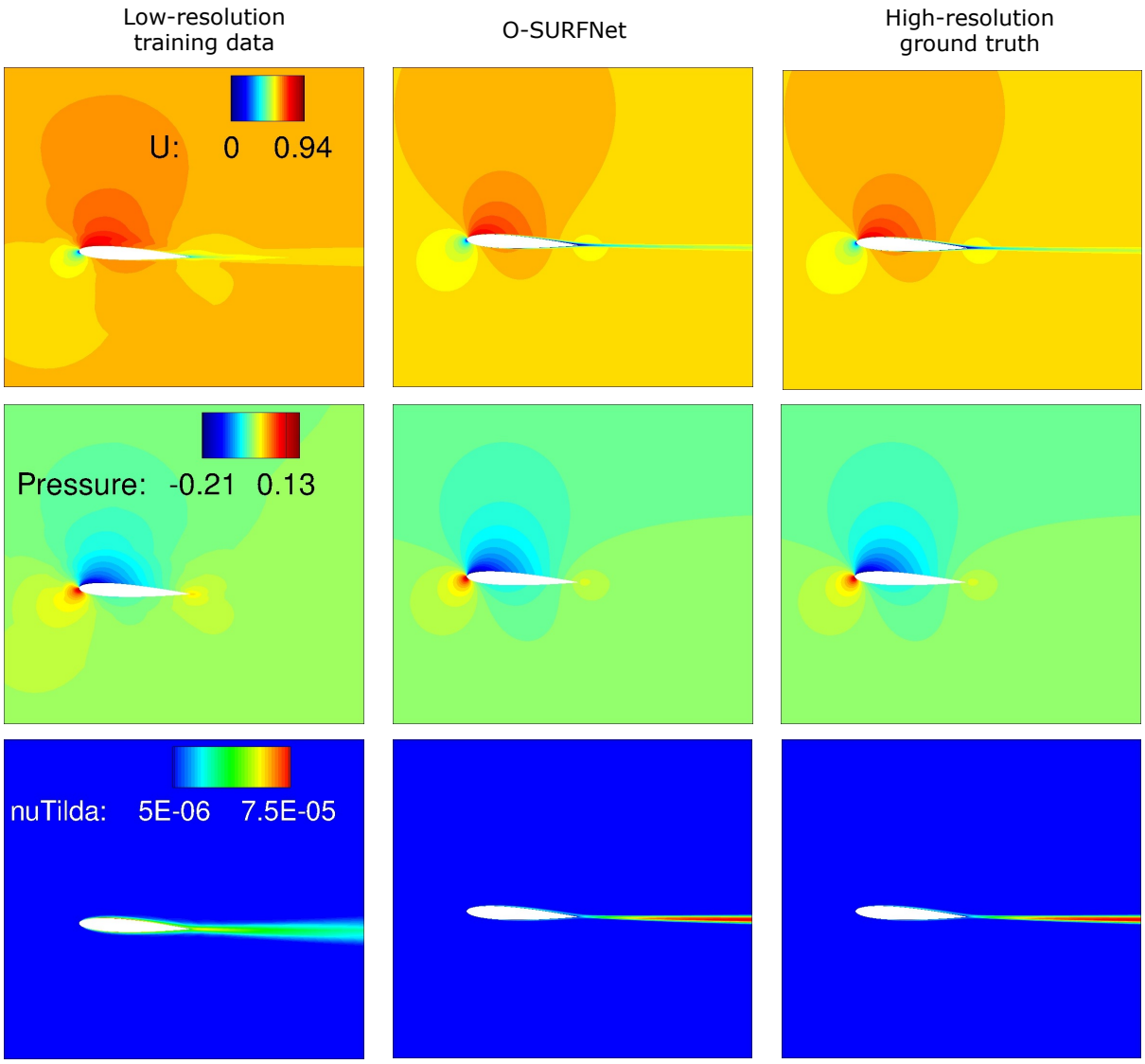


Figure 4.7: Velocity in $\mathrm{m\,s^{-1}}$ (top), kinematic pressure in $\mathrm{m^2/s^2}$ (middle), and modified eddy viscosity in $\mathrm{m^2/s}$ (bottom) around the NACA 0015 airfoil at $\theta = 3°$, $Re = 6 \times 10^5$. Comparison between the low-resolution (64×256) training data to train the coarse model (CM) (left); O-SURFNet's output after the refinement phase at 2048×2048 (middle), and the ground truth OpenFOAM's solution at 2048×2048 (right).

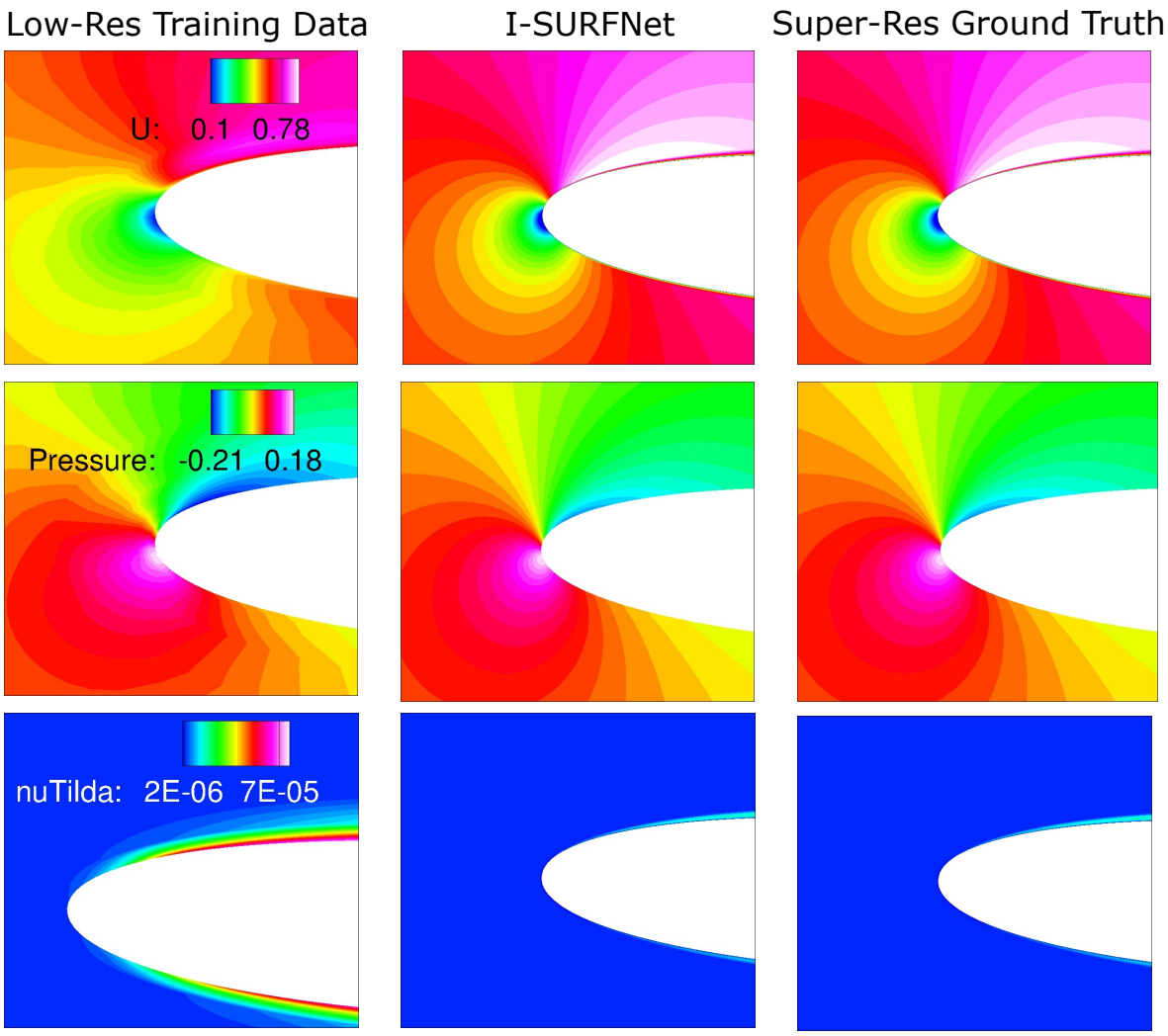


Figure 4.8: Detail of the velocity in $\mathrm{m\,s^{-1}}$ (top), kinematic pressure in $\mathrm{m^2/s^2}$ (middle), and modified eddy viscosity in $\mathrm{m^2/s}$ (bottom) at the nose of the nonsymmetric NACA 1412 airfoil at $\theta = 5°$, $Re = 6 \times 10^5$. Comparison between the low-resolution (64×256) training data to train the coarse model (left), I-SURFNet's output after the refinement phase at 2048×2048 (middle), and the ground truth OpenFOAM's solution at 2048×2048 (right).

By incrementally transferring with a step size of 2 up to 2048×2048, the performance gain drops to $1.7 - 1.8\times$. So, we conclude that to achieve oracle-level accuracy and performance, the ideal step size for ITL is 1.

4.3.3 SURFNet versus oracle

We now compare SURFNet with the oracle or baseline model (BM) at 256×256 and 512×512 target discretizations. We define BM as conducting full-scale data collection *and* training at the higher resolutions. Specifically, a dataset as large as the training dataset outlined in Table 4.2 collected at the tiny resolution (64×256) for pre-training the coarse model is collected for training the CNN at the two target discretizations. The CNN is trained using a learning rate of 1×10^{-4} and a batch size of 32 and 16, respectively. Figure 4.6 and Table 4.3 show that the loss on the validation dataset and speedup compared to the OpenFOAM physics solver achieved by ITL is similar to that of the BM. To understand how much time SURFNet saves with respect to the oracle, Table 4.5 compares the data collection and the training time of SURFNet with that of BM. SURFNet's data collection time is the sum of the time spent collecting low-resolution training data and the transfer datasets at higher resolutions. Similarly, the total training time includes pre-training the coarse model and the time spent fine-tuning at the target higher resolution.

In terms of data collection time, SURFNet takes 0.4 hrs (0.33 + 0.06) versus 1 hr for BM at 256×256. At 512×512, SURFNet's data collection time is 0.8 hrs (0.33 + 0.06 + 0.41) versus 3.86 hrs for BM. Training at 256×256 and 512×512 takes 2.56 hrs and 3 hrs for SURFNet versus 9.25 hrs and 38 hrs for BM, respectively. Overall, SURFNet reduces the combined data collection and training time by $3.6\times$ and $10.2\times$, respectively, while achieving similar performance gain and accuracy as BM. Note that the computational advantage of SURFNet increases with increasing resolution size. Moreover, these results highlight the

impracticability of performing exhaustive data collection and training at 1024×1024 and beyond, underscoring the impact and potential of transfer learning (specifically ITL) in enabling super-resolution of complex turbulent flows.

	SURFNet		**BM**	
	256×256	512×512	256×256	512×512
Data collection time for training	0.33	0.33	1	3.86
Data collection time for TL	0.06	0.41	-	-
Training time	2.5	2.5	9.25	38
TL time	0.06	0.5	-	-

Table 4.5: Comparison with the baseline model (BM) on data collection and training for reaching similar accuracy at 256×256 and 512×512 spatial resolutions.

4.4 Related work

Several recent approaches aim to find DL-based accelerators for turbulent flows with promising results. Maulik et al. [41] predict the eddy viscosity field, but not other flow properties such as the velocity. Thuerey et al. [42] use an encoder-decoder type of network but their approach does not account for the eddy viscosity field. Although they present real-time solutions, the geometry in the training and prediction stages are same (that is, airfoils) making the solution less generalizable. Like CFDNet, these early works are feasible only with low-resolution training data. They do not explore how to scale to larger resolutions. In [39], the authors accelerate fluid simulations by minimizing the divergence of the velocity field using an unsupervised method and in [54], the authors show that the proposed method is resolution invariant up to a 1024×1024 spatial resolution. Nevertheless, these works target Eulerian fluid simulations, which ignore viscous effects that are critical for practitioners. The applicability to turbulent flows remains unexplored.

There have been recent attempts to perform super-resolution for CFD and predict high-resolution flow fields. Some works [77, 90, 91] train a CNN as a low-resolution to high-resolution finite dimensional map. Therefore, they are not resolution invariant and require to restart the training of the model at each desired target resolution. *Mesh-free* methods [50, 51, 53, 92] achieve resolution-invariance across several resolutions with a single training of the NN. In [92], the authors introduced NOs. This class of approaches maintain accuracy up to 421×421 [50] and 241×241 [51]. However, they suffer from two disadvantages. First, they perform extensive data collection at their target resolution to train the NN. Second, the coarse-grid input data is downsampled from the collected high-resolution data. It remains unexplored how the models would perform with coarse-grid data as input. In [53], the authors achieve resolution-invariance to higher resolutions (such as 512×128) with a CNN-based model. Nevertheless, it suffers from the same two disadvantages. SURFNet scales to a 2048×2048 spatial resolution via transfer learning, requires $15\times$ less data, and trains the main model using *true* coarse-grid data.

4.5 Conclusions of SURFNet

We presented SURFNet, a transfer learning-based framework that can eliminate the need to collect large training datasets at high resolutions to account for fine-scale physical phenomena. This computational efficiency enables SURFNet to achieve oracle accuracies while significantly reducing the size of the training dataset by $15\times$, consequently reducing the combined data collection and training time by $3.6\times$ and $10.2\times$, respectively at 256×256 and 512×512 grid sizes. These promising results show that transfer learning is a valid technique for model re-use in fluid mechanics.

However, SURFNet suffers from a disadvantage: it still requires high-resolution data, even if the number of samples is small. Training with large input sizes, such as 1024×1024, impacts

the scalability of the method. For example, in our SURFNet experiment, the maximum batch size allowed by the memory limit of the GPU when transfer learning[4] at 1024×1024 was 2 underutilizing the GPU resources.

We observe that a high-resolution solution in the entire domain might not be necessary. A popular technique in CFD is AMR, which makes use of the fact that high numerical accuracy is only required in regions of the flow that present complex flow physics, such as the boundary layer or separated/detached flow. Smoother areas, such as the freestream (that presents very subtle changes), do not need a higher accuracy than that offered by a coarse mesh. Hence, this *non-uniform* approach allows reaching higher mesh resolutions with the same computational resources than those achieved with *uniform* super-resolution.

All works presented in Table 4.1 for super-resolution with DL perform *uniform* super-resolution. Even though mesh-free approaches can do an imbalanced allocation of collocation points, their methods do not inherently distinguish between different regions of the domain and target a higher resolution solution in the entire domain. Hence, designing a DL algorithm that only refines certain domain areas would be beneficial for scalability and performance.

In the next chapter, we present ADARNet, a DL algorithm for *non-uniform* super-resolution, where the network focuses on different regions or *patches* of the domain via a *scorer* module. Then, each region maps to a different target resolution.

[4]We trained using a mixed-precision strategy, where the forward pass is computed using 16-bit double precision and the gradients are computed using 32-bit double precision to avoid gradient underflow.

Chapter 5

ADARNet: DL predicts AMR

SOTA DL models for super-resolution, described in Table 4.1, have shown promise as domain-agnostic models and real-time predictors that generalize to a broad set of flow configurations and conditions. However, they **all** suffer from the fundamental limitation of performing *uniform* super-resolution, that is, *every* pixel of the input low-resolution image is refined to the target high resolution. As a result, SOTA [53, 90, 93] methods for super-resolution need higher computational resources - increased inference times and memory requirements - since the target high-resolution solution is output in the entire physical domain. Figure 5.1 shows the maximum allowable batch size with increasing target spatial resolution of these methods. On a 16GB NVIDIA V100 GPU, these approaches do not allow more than two samples per batch during inference at high spatial resolutions, such as 1024×1024, where more aspects of the physical phenomena can be modeled. This severely limits the deployment of DL methods for accelerating design space exploration in CFD.

Spatially uniform outputs are computationally inefficient for two other reasons. First, they can under-resolve areas with complex flow features and over-resolve regions with smooth fluctuations in the flow properties. It is critical to capture this versatility for complex systems

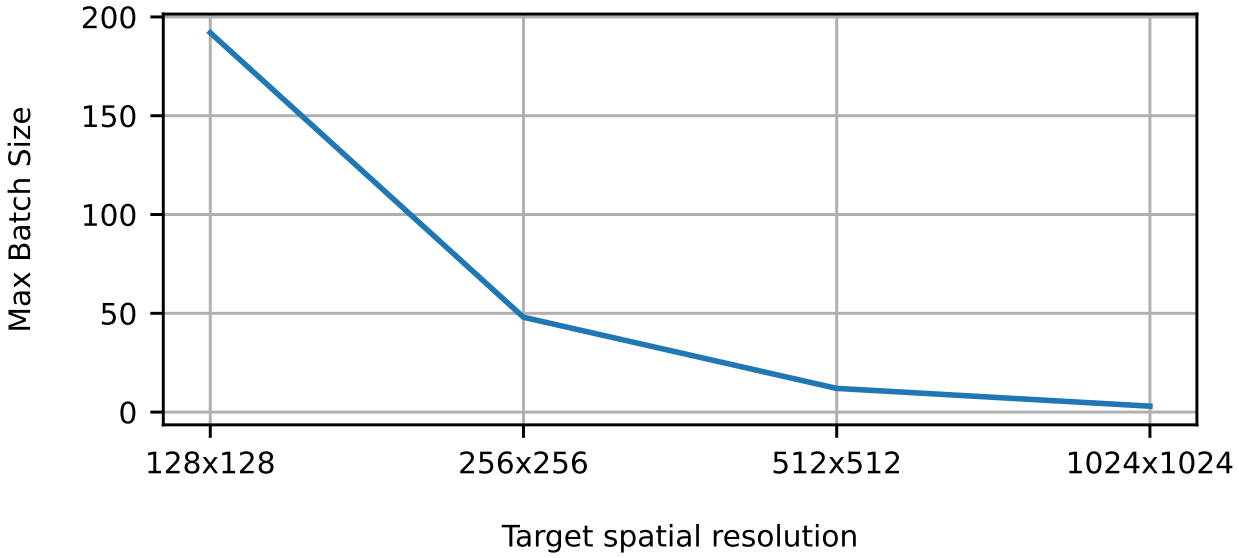


Figure 5.1: Maximum allowable batch size during inference at different target spatial resolutions for SOTA [93] DL super-resolution methods on a 16GB NVIDIA V100 GPU with 16-bit floating point precision.

with significant variations in their local solutions, such as turbulent flows. Second, current uniform super-resolution approaches need to know the target resolution *a priori*. As a result, they require many high-resolution labels at that specific resolution. Hence, they need to rely either on publicly available datasets or on performing data collection. Since the resolutions of the publicly available datasets are very limited, the majority of works [53, 90–92] end up performing computationally challenging high-resolution data collection.

Due to the large scale of many applications, it is often infeasible to solve the problem on a uniform mesh to achieve the desired accuracy. For this reason, traditional numerical solvers do not refine the entire mesh but do so adaptively, refining only regions of strong flow variability for scalability and performance - a method commonly referred to as AMR [55, 94]. However, traditional AMR methods in CFD suffer from two fundamental limitations. First, a high degree of user intervention in the refining/coarsening decisions: these decisions are based on heuristics that require problem-specific knowledge and do not generalize well. Second, the mesh is refined iteratively, requiring more compute time and memory than direct methods.

In this chapter, we tackle all these challenges and present ADARNet, a novel DL-based **ADA**ptive mesh **R**efinement framework for non-uniform super-resolution. ADARNet takes as input a low-resolution flow field and outputs, in one-shot, its final non-uniform high-resolution solution, as seen in Figure 5.2.

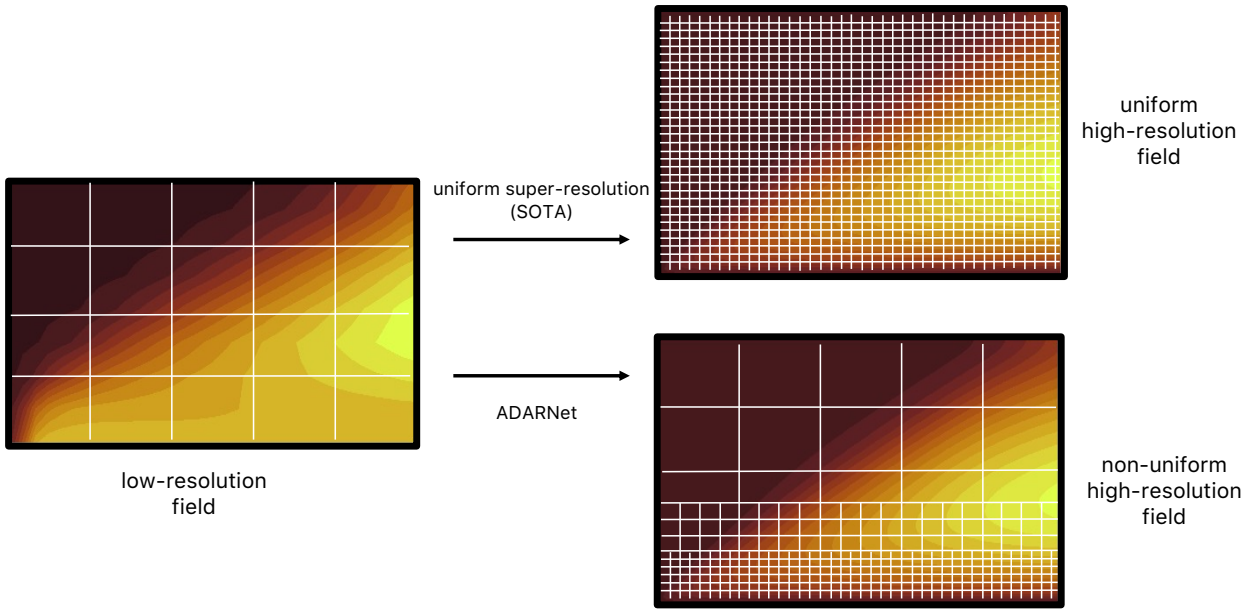


Figure 5.2: Current DL algorithms for super-resolution output the solution on a uniform fine mesh (top). Our objective with ADARNet is to predict a spatially non-uniform output where only areas that require higher accuracy are refined (bottom). Hence, ADARNet requires less compute time and memory resources while achieving the target accuracy.

Since only regions in areas that present complex flow phenomena are refined, it requires less computational resources. This enables larger batch sizes during inference at high spatial resolutions while reaching the target accuracy compared to SOTA methods for super-resolution. ADARNet distinguishes between different regions of the domain by dividing the input low-resolution flow field into fixed-size *patches*, adaptively increasing or maintaining the spatial resolution of each patch, and predicting a non-uniform high-resolution flow field. We present ADARNet as an end-to-end DL-physics solver framework where the non-uniform output field from the model inference is input to a traditional physics solver that drives the solution to convergence. As a result, ADARNet meets the same convergence guarantees as AMR solvers which is critical for practitioners [41, 79, 93].

5.1 ADARNet: DL for non-uniform super-resolution

Our objective is three-fold. First, to predict fine-grid turbulent flows from their coarse-grid counterpart only in the regions of interest. Second, to design a DL algorithm for AMR where these areas to refine are identified with the least possible user intervention. Third, to output a solution that meets the same convergence guarantees as classical AMR solvers.

In this section, we present ADARNet, a novel DL framework for adaptive super-resolution. We first describe in detail the NN architecture and then, present our semi-supervised learning approach that leverages a hybrid loss function. Finally, we outline the end-to-end framework, which reconstructs a non-uniform high-resolution flow field while reaching the same convergence as SOTA AMR solvers.

5.1.1 NN architecture

We choose a DNN for the task of non-uniform super-resolution. The input to the DNN is a low-resolution flow field, which is divided into fixed-size regions or *patches*. The output is a non-uniform resolution flow field, where high-resolution is given only at specific *patches* of the domain. The RANS equations with the SA model predict four main flow variables – mean x-velocity (U), mean y-velocity (V), the mean kinematic pressure (p), and the eddy viscosity ($\tilde{\nu}$). Therefore, the input low-resolution flow field consists of a four-channel tensor image where each channel represents the values of one flow variable in the entire computational grid. The DNN scores, ranks (or bins), and predicts the target resolution of each patch. Figure 5.3 illustrates the architecture composed of a *scorer* network, a *ranker*, and a *decoder* network.

Scorer. The low-resolution flow field is first input to the *scorer* network. This is a trainable network whose goal is to score each patch of the low-resolution image via its 2D spatial

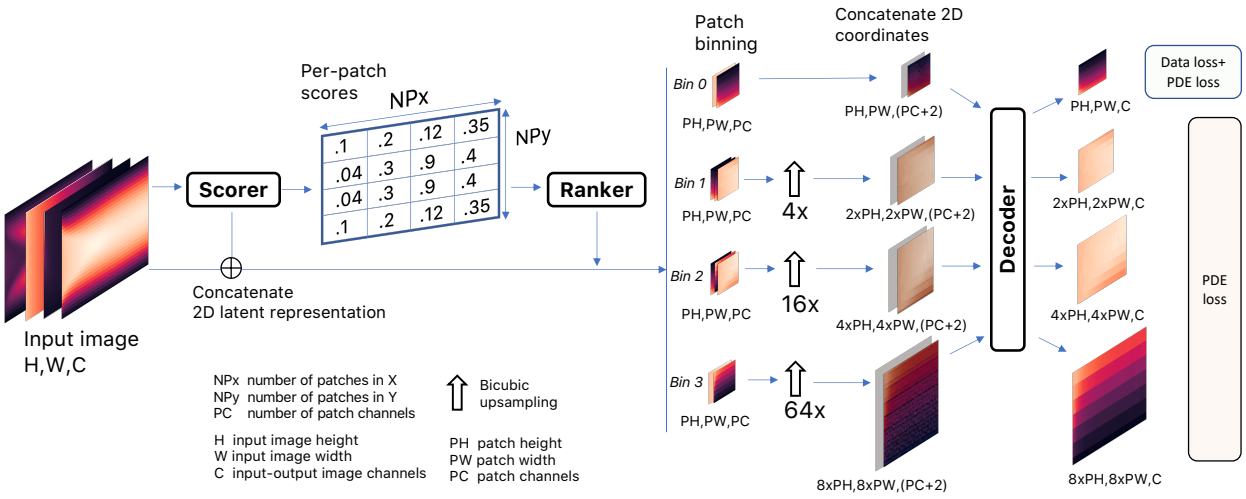


Figure 5.3: ADARNet's DNN. The input is a four-channel low-resolution image where each channel represents one flow variable. The low-resolution image is first input to the *scorer*, which divides it into patches and outputs the score of each patch. The *ranker* uses these scores to assign each patch to a bin, which is subsequently upsampled using a bicubic interpolation to its target resolution. Then, the upsampled patches are concatenated with their coordinates. Finally, the *decoder* maps this upsampled, intermediate representation to the final values of each patch. ADARNet's DNN's output is multiple, consisting of a list of four-channel images at different spatial resolutions.

latent representation, as illustrated in Figure 5.4. This network is inspired by the work of Cordonnier et al. [95] that use a similar network for finding salient patches from the input image for classification.

The *scorer* network consists of a shallow CNN followed by a maxpooling layer and a softmax layer. The first three convolutional layers extract an abstract representation of the low-resolution flow field. Their kernel size is $(5, 5)$ and the stride is 1. This overlap captures the spatial relationships between and among the input flow variables – a $(5, 5)$ kernel can capture both short and long-range dependencies and a stride of 1 maintains the spatial dimensionality. After the first three convolutional layers, we apply a single-filter convolutional layer to squeeze and encapsulate the extracted abstract spatial information in a single-channel image. This image is a 2D latent representation of the spatial dependencies in the variables of the low-resolution flow field and plays a key role in determining the scores of each patch. This single-channel image is input to the maxpooling layer that splits the domain into

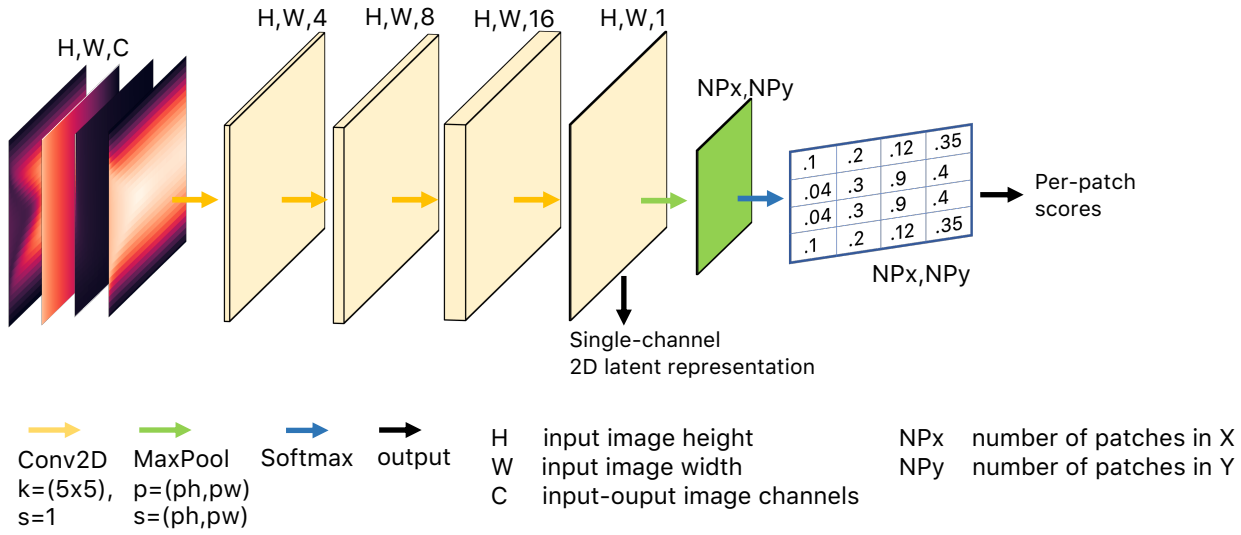


Figure 5.4: The *scorer* network. It consists of four convolutional layers followed by maxpool and softmax layers. The first three convolutional layers extract a single-channel 2D latent spatial representation from the input low-resolution flow field. This 2D latent representation is used to obtain the scores of each patch via a maxpooling and a softmax layer. It is also output and concatenated with the original low-resolution image.

$NP_x \times NP_y = N$ patches – where NP_x is the number of patches in the horizontal direction, NP_y is the number of patches in the vertical direction, and N is the total number of patches. The pool size and the stride are both (ph, pw), where ph is the height of the patch and pw is the width of the patch. Hence, each value in this image represents the non-normalized score of each patch. The softmax layer normalizes these scores to a $0-1$ scaled probability distribution. The *scorer* network's output is twofold: the scaled scores and the 2D latent representation.

Before passing the scores to the *ranker*, we concatenate the 2D spatial latent representation with the low-resolution flow field. This is motivated by two reasons. First, the latent representation already contains spatial correlations from the low-resolution flow field. Second, it allows to dynamically change the scores of the patches since the *scorer's* weights are propagated during the backward pass of the training process. Hence, the low-resolution flow field becomes a five-channel image (in Figure 5.3, $PC = 5$). Finally, we pass the scores from the *scorer* to the *ranker*.

Ranker. This is a non-trainable module that tracks the score and the ID of each patch. The *ranker* locates each patch in the new five-channel low-resolution flow field, isolates it from the rest of the image, and places it in a bin according to its score. We refer to this process as *binning*, and we illustrate in Figure 5.3. Binning consists of splitting the $0-1$ range of values of the scores into b bins uniformly.[1] For instance, if $b = 2$, the first bin consists of patches with scores between $0 - 0.5$, and the second bin with scores $0.5 - 1$. The *ranker* plays a significant role in determining the final resolution of each patch: training consists of mapping the highest-scored patches to the highest target resolution, and as scores gradually decrease, so does the target resolution of the patch. The patches with the lowest scores remain low-resolution.

After the binning finishes, we perform two additional steps before passing the patches to the *decoder*. First, we upsample (refine) each patch to its target resolution using bicubic interpolation, as seen in Figure 5.3. Then, we concatenate the 2D coordinates to each patch needed to compute the gradients of the flow variables using automatic differentiation [96]. This leads to a seven-channel image (in Figure 5.3, $PC + 2$). Once we upsample and concatenate the coordinates to the patches, it is input to the decoder.

Decoder. The goal of this trainable network illustrated in Figure 5.5 is to reconstruct the high-resolution solution of each patch. Each patch in each bin passes through the same decoder, which is shared among resolutions. Note that the patches placed in the low-resolution bin also passes through the decoder. The choice of a shared decoder is motivated by two reasons. First, we have a smaller number of learnable parameters compared to a separate decoder for each resolution. Thus, we stress ADARNet's ability to recover different resolutions for different patches with a lower computational cost. Second, the low-resolution patches have not been upsampled and the decoder can extract the true spatial correlations between

[1] Another approach is to bin non-uniformly, which would change each patch's placement and their final resolution, giving more or less importance to higher-scored patches. However, in this thesis we only explore uniform binning.

the flow variables and coordinates in those patches. We expect this to help in recovering the true values of the high-resolution patches.

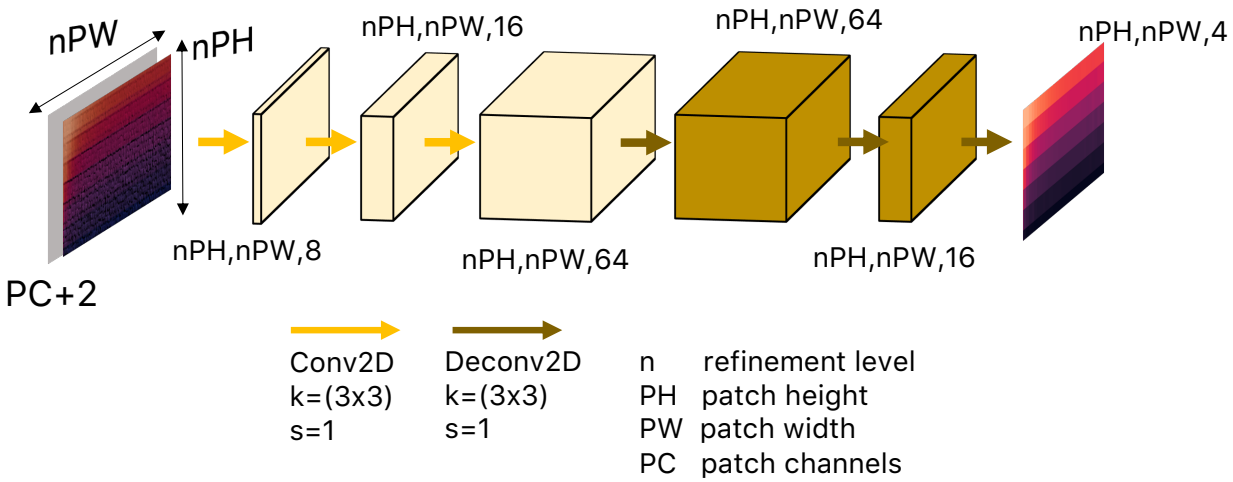


Figure 5.5: The *decoder* network. The input to the decoder is the intermediate patch representation concatenated with the 2D coordinates at its target resolution. This network consists of 3 convolutional layers followed by 3 deconvolutional layers that reconstruct the final values of the patch at its target resolution. The output is, therefore, a four-channel image, where each channel is the value of a flow variable.

This network is a 6-layer network; 3 convolutional layers followed by 3 deconvolutional layers. The number of filters is 8, 16, 64, 64, 16, 4; the kernel size is $(3, 3)$, and the stride is 1. The choice of this architecture is twofold. First, recent works have successfully leveraged similar architectures for flow super-resolution [77, 93]. Second, the convolutional layers aim at extracting a deep, abstract representation used by the deconvolution layers to reconstruct the high-resolution output. We use a stride of 1 to not lose any spatial information.

The decoder's output consists of a list of patches. Each patch is a four-channel image, and each channel represents the values of the four flow variables (U, V, p, and $\tilde{\nu}$) at steady-state at their new spatial resolution. Each patch in the list can have a different spatial resolution. This list is then passed to the loss function.

5.1.2 Loss function

Recent DL-based super-resolution approaches have leveraged data-only loss functions [90, 93], where CFD simulations generate the ground truth labels. Other approaches use a combination of data and PDE residual loss function [53, 77], where the governing equations are imposed in the loss function. We adhere to the latter practice for a couple of reasons. First, it is physically and numerically inconsistent to merge CFD-originated data from different spatial resolutions. Second, we do not train with ground truth data from AMR solvers because that would make the network learn the solver's heuristics that have a high degree of user intervention. The goal is for ADARNet to make its own refining decisions to obtain a DL-based model for AMR. Equation 5.1 shows our loss function.

$$\mathbb{L} = \frac{1}{np \cdot nc \cdot fv} \sum_{i=1}^{np} \sum_{j=1}^{nc} \sum_{k=1}^{fv} \|\hat{y}_{ijk} - y_{ijk}\|^2 + \frac{\lambda}{NC \cdot ne} \sum_{i=1}^{NC} \sum_{j=1}^{ne} \|R_j(i)\|^2 \tag{5.1}$$

Our loss function $\mathbb{L}$ is formed by two parts. The first term in the right hand side of Equation 5.1 is the data loss. We take the MSE of the prediction of each flow variable (fv) at each cell (nc) of the low-resolution patches only (np) with the ground truth data obtained with the physics solver. The second term in the right hand side is the L2 norm of the residual (R) of each PDE (ne) for all cells (NC) of the output image, belonging to both low- and high-resolution patches. We impose the continuity equation and the two conservation of momentum equations (hence, $ne = 3$), described in Equations (2.1) and (2.2). The gradients of the variables are computed through automatic differentiation [96]. To constraint the PDE residual of the high-resolution patches, we downsample them using bicubic interpolation to the lowest resolution and match the ground truth data in the downsampled space [77]. With this semi-supervised learning formulation, we avoid high-resolution labels, an advantage over SOTA super-resolution methods that require expensive high-resolution training

data [50, 51, 53, 92].

5.1.3 End-to-end framework

Once the network is trained and calibrated, we use it to predict the non-uniform high-resolution flow field of a new problem. However, this prediction has an approximation error and might not satisfy the same convergence constraints as traditional physics solvers with PDE residual values close to the machine round-off errors, which is critical for many practitioners. We correct this by augmenting the DNN's prediction with the physics solver [41, 79, 93].

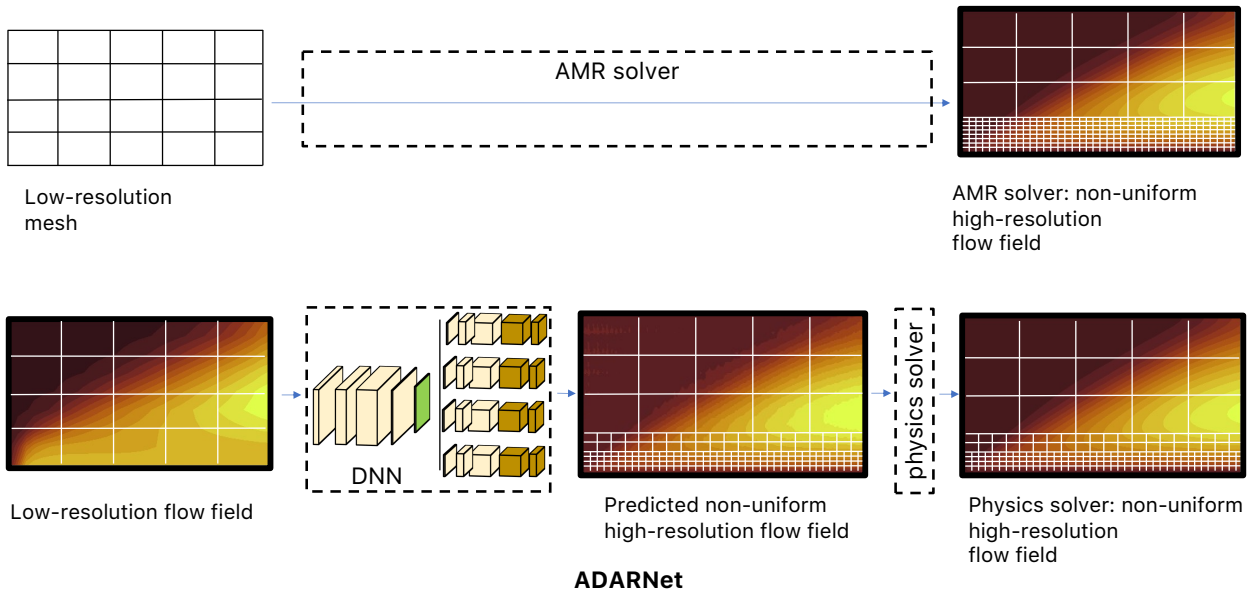


Figure 5.6: Top: Traditional AMR solver simulation. Bottom: ADARNet framework. After performing non-uniform super-resolution with the DNN, we feed the output field into the physics solver, which takes the inferred solution to convergence.

Figure 5.6 illustrates ADARNet's end-to-end framework compared with the traditional AMR solver. In ADARNet, the low-resolution flow field is input to the DNN. After inference, we feed the DNN's non-uniform output into the physics solver, impose the boundary conditions that well-pose the problem, and let the physics solver drive this inference to convergence.

Note that the physics solver does not do any further refinement or coarsening. The final discretization is an output of the DNN.

As a result, we obtain a solution that satisfies the same convergence constraints as traditional numerical methods. Since we anticipate the DNN's inference to be close to the final solution, convergence is accelerated. Section 5.3 empirically evaluates the performance of ADARNet against both classical AMR solvers and SOTA DL models.

5.2 Experiment setup

In this section, we present the methodology to train and evaluate ADARNet. We first present the dataset to train and validate ADARNet along with the training/testing setup, parameters, and results. Then, we describe the physics solver and the traditional AMR heuristics used for comparison. We use the steady incompressible RANS problems for our non-uniform super-resolution task.

5.2.1 Dataset overview and flow description

We gather low-resolution data from three well-known canonical flows for training the DNN in Figure 5.3. The resolution for this dataset is 64×256, since it is a common resolution for low-resolution solutions for all our training cases [1].

Turbulent flow in a channel. 2D channel flow has been widely studied in the literature [97]. A common strategy to evaluate channel flow is to vary the input velocity to the channel. This is the same as varying the Re since $Re = \frac{UL}{\nu}$, where U is the input velocity to the channel, L is the diameter of the channel, and ν is the laminar viscosity of the problem. Here, we adhere to this practice and vary the input velocity to the channel to collect 10000

samples. Specifically, we collect 300 samples from $Re = 2 \times 10^3$ (when turbulent effects start to appear [1]) to $Re = 2.3 \times 10^3$, and then, 9700 more samples from $Re = 2.7 \times 10^3$ to $Re = 1.35 \times 10^4$. We leave $Re = 2.5 \times 10^3$ and $Re = 1.5e4$ as our test cases. Section 5.3 presents a more in-depth discussion of the selection of the test cases. The physical domain of the channel flow is a diameter of 0.1 meters and a length of 6 meters so we find fully developed flow. The inlet is at the left and the outlet at the right. The top and the bottom are both walls and hence have the no-slip boundary condition. The velocity boundary conditions are uniform inlet at the inlet, no-gradient at the outlet, and 0 at the walls. The pressure boundary conditions are no-gradient at the inlet, 0 uniform at the outlet, and no-gradient at the walls. The modified eddy viscosity boundary conditions are 3×(laminar viscosity) at the inlet, no-gradient at the outlet, and 0 at the walls. The laminar viscosity is set to 1×10^{-4} m^2/s .

Turbulent flow over a flat plate. Flat plate is also a canonical flow, part of the wall-bounded flows family, used to study the boundary layer in both laminar and turbulent conditions [1]. By varying the incoming velocity we collect 10000 samples. For flat plate, incompressible turbulent effects do not appear [1] up until $Re = 1.35 \times 10^5$ and scale up to $Re = 5 \times 10^6$. We collect 2000 samples from $Re = 1.35 \times 10^5$ to $Re = 2 \times 10^5$ and another 8000 additional samples from $Re = 3e5$ until $Re = 1.1e6$. We leave $Re = 2.5 \times 10^5$ and $Re = 1.35 \times 10^6$ as test cases. The physical domain of the flat plate case is a height of 0.2 meters and a length of 10 meters, as found in different benchmarks. The boundaries are a wall at the bottom (the flat plate), symmetry at the top, an inlet at the left, and an outlet at the right. The velocity boundary conditions are uniform inlet at the inlet, no-slip condition at the bottom wall, no-gradient both at the outlet. The pressure boundary condition is no-gradient at the inlet and bottom wall, and 0 at the outlet. The modified eddy viscosity boundary conditions are 3×(laminar viscosity) at the inlet, 0 at the bottom wall, and no-gradient at the outlet. The laminar viscosity is set to 1×10^{-4} m^2/s.

Turbulent flow around ellipses. External aerodynamics simulations are relevant for aerospace industrial applications. We gather low-resolution solutions from flow around ellipses. In real scenarios, different geometries at a variety of flow conditions are explored. Our training data consists of 10000 samples of flow around different ellipses at different flow conditions. Figure 5.7 shows these configurations.

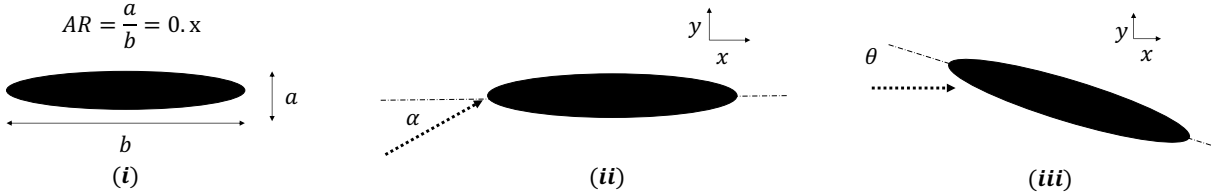


Figure 5.7: Sketch of the (i) ellipse aspect ratio, (ii) angle of attack α, and (iii) pitch angle θ.

The training data is obtained by changing the aspect ratio of the ellipses: 0.05, 0.07, 0.09, 0.1, 0.15, 0.2, 0.25, 0.35, 0.55, and 0.75. Each of these ellipses is simulated under 5 different flow conditions by changing randomly the angle of attack α and the pitching angle θ between -2 and 6 degrees. We collect all of these configurations at 200 different Re numbers between 5×10^4 and 9×10^4. We select flow around a cylinder at $Re = 1 \times 10^5$, flow around a symmetric airfoil (NACA0012) at $Re = 2.5 \times 10^4$, and flow around a non-symmetric airfoil (NACA1412) at $Re = 2.5 \times 10^4$. The physical domain of the ellipse/cylinder/airfoil cases consists of a solid body of chord (c) 1 meter, and the far-field limit is located 30c from the tip and tail of the solid body (O-grid type of mesh). The velocity boundary conditions are no-slip at the wall and uniform velocity at the far-field; the pressure boundary conditions are no-gradient at the wall and uniform 0 in the far-field; the modified eddy viscosity boundary conditions are 0 at the wall and uniform 3×(laminar viscosity) in the far-field, where the laminar viscosity is 1×10^{-4} m^2/s uniform in the entire domain.

The total training set size is composed of 30000 samples, 10000 from each canonical flow. From this training dataset, 27000 samples are used for training the DNN and 3000 samples are used for validation.

5.2.2 Training and testing setup

The methodology described in Section 5.1 allows for multiple combinations of parameters in addition to those in NN optimization. For instance, we can select different patch sizes or different number of bins (and therefore, different number of target resolutions). In this section, we explain our training design choices.

First, ADARNet's DNN's convolutional kernels of size 5×5 and 3×3 require a minimum input image size to extract relevant information. Therefore, we fix our patch sizes at $ph \times pw = 16\times16$, which leads to $N = 64$ total number of patches for each train/validation/test sample. Larger patch sizes (for instance, 32×32) do not offer enough granularity to make critical distinctions between regions of the flow. Second, we choose the number of bins $b = 4$, and hence four different target resolutions. Each target resolution refines the original low-resolution patch by $4^{(n)}\times$, where $n = 0, 1, 2, 3$. We choose $b = 4$ because not more than 4 levels of refinement is an extended practice in the AMR literature [98, 99] to avoid tiny computational cells. This also allows us to compare ADARNet with SOTA approaches that attempt $64\times$ super-resolutions. The patch size and the number of bins are the same at test time and during the evaluation of the results.

We implement the DNN using the Tensorflow 2.4 backend, and perform distributed training on four Tesla V100 GPUs connected with PCIe. After training the network with a batch size of 8, a learning rate of $1e-4$, no specific initialization, and using the Adam optimizer [100] for 350 epochs, the training and validation data and PDE residual loss for all equations reach a MSE of 9e-6. Note that the training of ADARNet's DNN is done on the Tesla V100 GPUs. However, ADARNet is implemented entirely on the CPU for a fair comparison with the AMR solver, which only supports CPU. Hence, both ADARNet and the AMR solver are executed on the CPU.

5.2.3 Physics solver and AMR solver

Once the network is calibrated, it is used for inference. Recall that the DNN's output is input into the physics solver to drive the flow field from inference to convergence. We use OpenFOAM's `pimpleFoam` [20] solver as the physics solver in this thesis. For the pressure, we use the `GAMG` solver, with a relative tolerance of 0.1, and the `GaussSeidel` smoother from OpenFOAM. For the velocity and the eddy viscosity, we use the `smoothSolver` with a relative tolerance of 0.1. The number of outer and inner correctors are set to 1, and the time scheme is set to steady-state.

As for the AMR solver to compare ADARNet with, we use the `dynamicMeshRefine` utility in OpenFOAM. This utility performs AMR as long as it is used together with `pimpleFoam` solver. The combination of the `pimpleFoam` solver with the `dynamicMeshRefine` utility forms the AMR solver used in this thesis. This solver is a feature-based AMR solver, which is the most popular method in the literature [55]. Therefore, it requires user intervention: for all of the test cases, we set the AMR solver to refine those areas where the gradients of the eddy viscosity are the highest, and the maximum level of refinement is set to 4. This heuristic is popular and works well for a wide range of problems, including our test problems [1].

Architecture and Libraries. All the OpenFOAM simulations are run in parallel on a dual-socket Intel Xeon Gold 6148 using double precision due to the lack of GPU support. Each socket has 20 cores, for a total of 40 cores. We use the OpenMPI implementation of MPI integrated with OpenFOAM v8 that is optimized for shared-memory communication. The grid domain is decomposed into 40 partitions using the integrated Scotch partitioner and each partition is assigned to 1 MPI process that is pinned to a single core. We set the `numactl -localalloc` flag to bind each MPI process to its local memory.

5.3 Results and discussion

After training and validating the network, we evaluate ADARNet's ability for non-uniform super-resolution on two different use cases, as outlined below:

I **Same geometry, different boundary conditions**. We use ADARNet to refine the low-resolution solution of flow on a geometry observed during training but at a different boundary condition. Here, our test flows configurations are channel flow and flat plate on interpolated (int) and extrapolated (ext) boundary conditions. For the former, we test on $Re = 2.5 \times 10^3$ (int) and $Re = 15 \times 10^3$ (ext). For the latter, we test on $Re = 2.5 \times 10^5$ (int) and $Re = 1.35 \times 10^6$ (ext).

II **Different geometry, different boundary conditions**. We stress the generalization capacity of ADARNet by finding the non-uniform high-resolution solution of flow around geometries unseen during training. We use the same network to predict the flow around a cylinder at $Re = 1 \times 10^5$, the flow around a symmetric NACA0012 [70] airfoil at$Re = 2.5 \times 10^4$, and the non-symmetric NACA1412 airfoil at $Re = 2.5 \times 10^4$, as seen in Figure 5.8.

For the described test cases, we first present the accuracy and correctness of ADARNet by comparing it, both qualitatively and quantitatively, with the traditional AMR solver (described in Section 5.2.3). Then, we present the performance analysis of ADARNet by showing (1) its speedup over the AMR solver and (2) the improved inference time and memory usage over SOTA NN models that perform uniform super-resolution. The baseline NN used for comparison is described in Section 5.3.2.

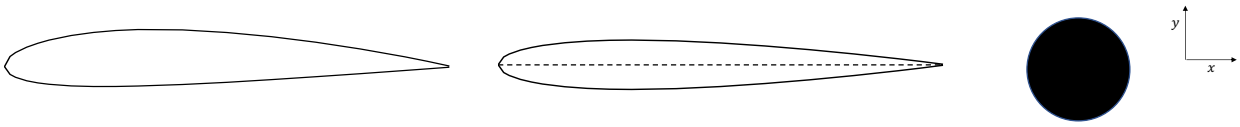


Figure 5.8: Non-symmetric NACA1412 airfoil (left), symmetric NACA0012 airfoil (center), and cylinder (right) as test geometries. These test cases stress the generalization capacity of ADARNet to unseen-during-training geometries.

5.3.1 Correctness and accuracy of ADARNet

Once the DNN is trained, we build the ADARNet framework and study its correctness and accuracy by comparing it with the traditional AMR solver.

We first conduct a qualitative evaluation by visualizing (a) the refined/unrefined areas and (b) the final flow field by the two algorithms. Because of the difference in their inherent heuristics (ADARNet follows an optimization process containing different flow cases while the AMR solver follows user-given heuristics as explained in Section 5.2.3), we do not expect the exact same output. However, the qualitative results evaluate whether ADARNet can act as an AMR surrogate for multiple flow problems resulting from a single training.

After, we present a quantitative comparison between the two using a grid convergence study [101]. Recall that both ADARNet and the AMR solver solve the same problem. We impose the same strong-form boundary conditions in the fluid domain, which well-pose the problem and guarantee uniqueness [89]. The only metric that changes between the two is the mesh, and therefore, they will present different discretization errors. However, these discretization errors reduce as we increase the resolution of refinement and global quantities tend to converged solutions [1]. Hence, to evaluate the quality of ADARNet's inferred mesh, we compare the solution from both ADARNet's mesh and the AMR solver mesh as we increase the required levels of refinement. Both meshes are refined $4^n\times$ gradually, from $n = 0$ to $n = 3$. Then, we report the value of specific quantities of interest (QoI) at steady-state. The choice of the QoI follows the CFD literature [1].

Qualitative results

Figure 5.9 shows the refined/unrefined results for five test cases: channel flow at $Re = 2.5 \times 10^3$, flat plate at $Re = 1.35 \times 10^6$, cylinder, and the two airfoil test cases. Figure 5.9 shows ADARNet's predicted mesh (left) and the AMR solver's output mesh (right). ADARNet splits the domain into 64 16×16 patches and we show the output resolution (with respect to the coarse resolution) of each patch. Because the AMR solver allows more granularity as it performs mesh refinement on a per-cell basis, the domain is divided into smaller (4×8) patches[2]. At the borders of each test case, we show the physical boundaries of each case which play a key role in determining the areas where both algorithms refine the mesh.

We make three main observations. First, ADARNet can distinguish between boundary conditions. For the channel flow case (first row in Figure 5.9), ADARNet refines the fluid areas close to both the upper and the lower wall, whereas, for the flat plate case (second row), it refines the areas close to the wall but leaves the outer regions (outlet/freestream) at low resolution. Second, ADARNet respects the symmetry of the problem, as we observe in the channel flow case (first row of Figure 5.9) and in the symmetric airfoil case (fifth row in Figure 5.9). Third, ADARNet's fine/coarse regions are in excellent agreement with those of the AMR solver for the channel flow, flat plate, and airfoil cases. This agreement in the cylinder case is also notable. For instance, ADARNet refines the region of the flow from the back of the cylinder to the outlet (that is, the wake behind the cylinder). However, we observe some discrepancy in the back-bottom-front-top region. The front-bottom-back and front-top-back regions (which refer to the entire solid boundary of the cylinder) require a higher resolution from ADARNet. This difference, together with the channel flow and flat plate results, indicates that the DNN is refining those areas with higher values of the gradients for *all* fluid variables, which take place in solid wall boundaries [1]. This is opposed

[2]We do not show per-cell refinement as too many cells are created to offer good visualization. However, 4×8 patch sizes have been found optimal for both gathering cells with equal levels of refinement and visualization quality.

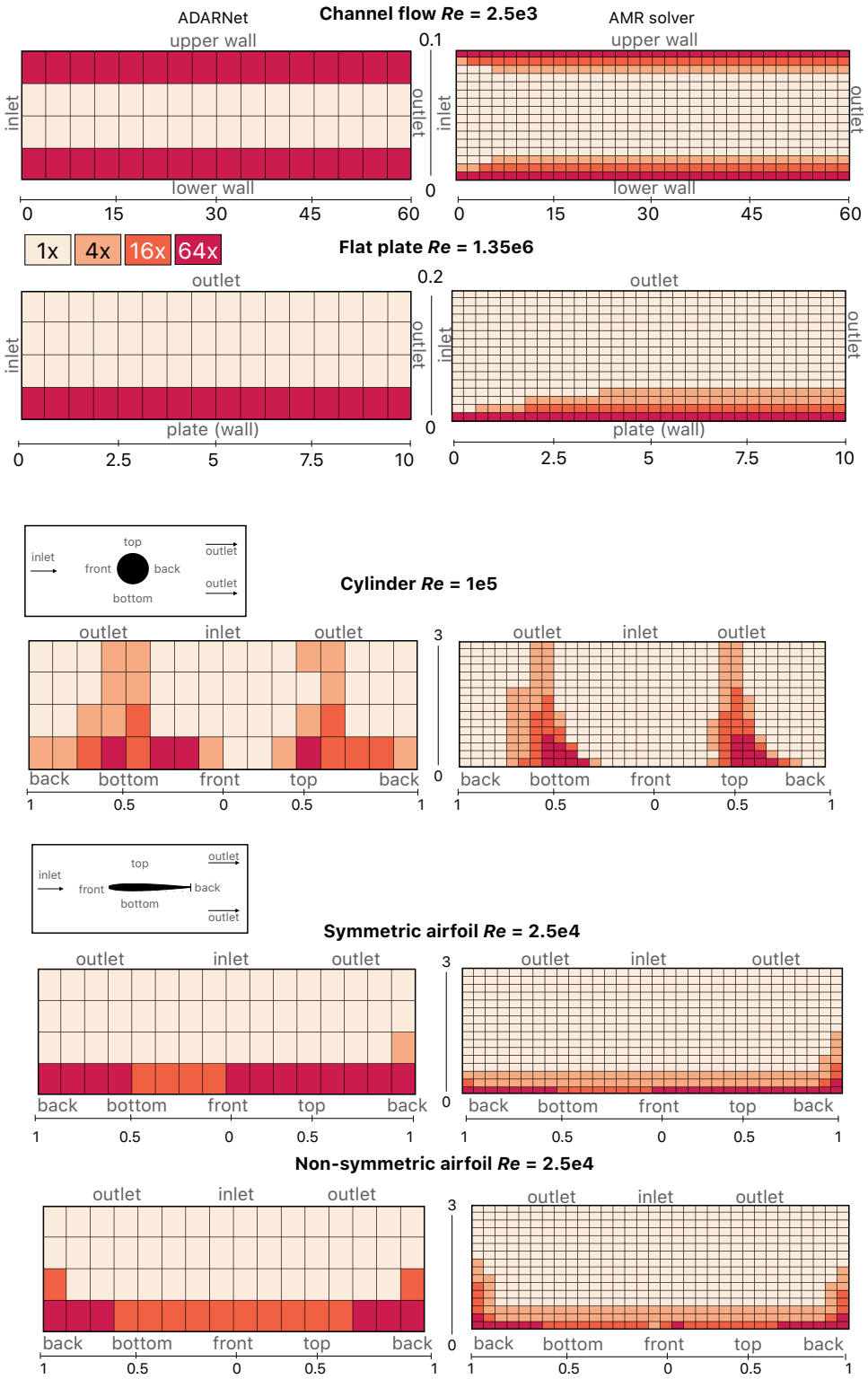


Figure 5.9: Per-patch fluid domain and the level of refinement of each patch for all our test cases. First row: channel flow at $Re = 2.5 \times 10^3$. Second row: flat plate at $Re = 1.35 \times 10^6$. Third row: cylinder at $Re = 1 \times 10^5$. Fourth row: symmetric NACA0012 airfoil at $Re = 2.5 \times 10^4$. Fifth row: non-symmetric NACA1412 airfoil at $Re = 2.5 \times 10^4$. We compare ADARNet's prediction (left) versus AMR solver's output (right). Both axes are in meters.

to the AMR solver's heuristic, that focuses only on areas with high gradients of the eddy viscosity.

During the calibration of the DNN, it is key to balance both components of the loss function - the data loss and the PDE residual loss (described in Section 5.1.2) so neither dominates the other. In our experiments, we observe the best predictive results at $\lambda = 0.03$, which yields a balanced contribution of each component of the loss. The data and PDE residual loss reached a value of 9×10^{-6} for both the training and the validation samples. During training, we scale the value of the variables between 0 and 1 for learning stability purposes. However, we can not scale the value of the gradients found by automatic differentiation because this would result in inconsistent PDE residual loss. These gradients reach higher absolute values than those of the data, especially in areas of the flow with higher variability, and hence get the attention of the MSE loss function. Moreover, this also allows ADARNet to refine the back-outlet area, where the wake region of the flow after the cylinder meets the freestream (outlet) and we find a high gradient of the eddy viscosity. The difference in the refining patterns between the cylinder and the airfoil case is that the former presents flow separation from the wall boundary that generates a wide wake region, whereas in the airfoil case the flow remains attached to the solid.

Figure 5.9 also shows that in the channel flow, flat plate, and airfoil test cases, the AMR solver reduces the refinement level gradually as we increase the distance from the wall boundary. Instead, ADARNet infers the maximum level of refinement in the patches close to the wall and does not show this gradual reduction. This is due to the maxpooling layer in the design of ADARNet's *scorer* network (see Section 5.1.1). Recall that the maxpooling layer chooses the highest score present in the 16×16 region defined by the patch. Choosing a maxpooling layer over an average pooling is a desired conservative approach. Since an entire patch shares a resolution in ADARNet, it is advantageous to choose the highest required resolution even if only few cells within a patch require it for accuracy. Figure 5.9 shows that ADARNet and

the AMR solver have inherently different heuristics for mesh refinement/coarsening and do not produce the same mesh - as expected. However, both are in excellent agreement in their steady flow field prediction, as we qualitatively observe in Figures 5.10, 5.11, 5.12, 5.13, 5.14.

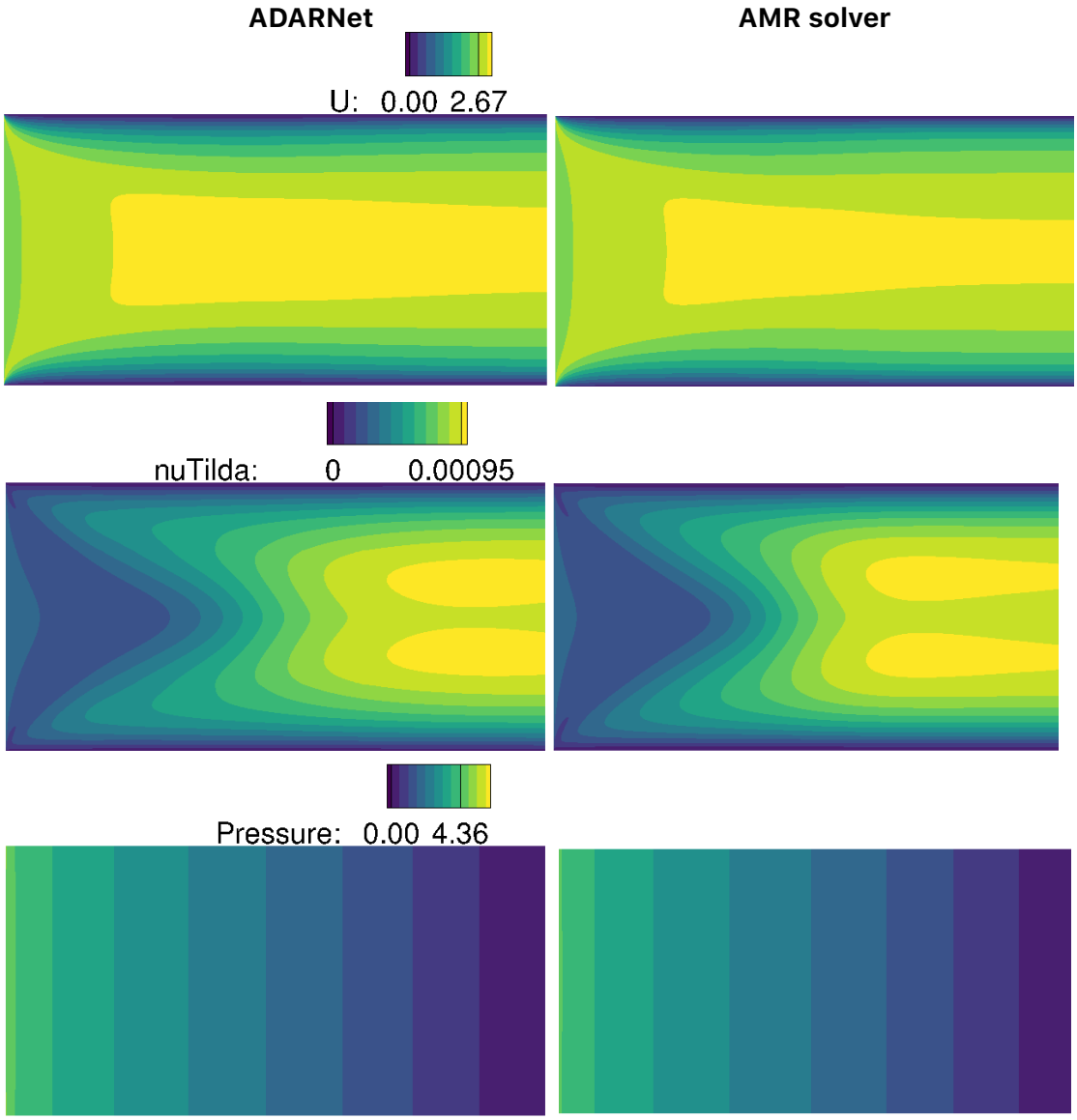


Figure 5.10: Velocity in $\mathrm{m\,s^{-1}}$ (top), kinematic pressure in $\mathrm{m^2/s^2}$ (bottom), and modified eddy viscosity in $\mathrm{m^2/s}$ (middle) for channel flow at $Re = 2.5 \times 10^3$. Comparison between ADARNet's result (left) and the AMR solver result (right) for $b = 4$ levels of refinement.

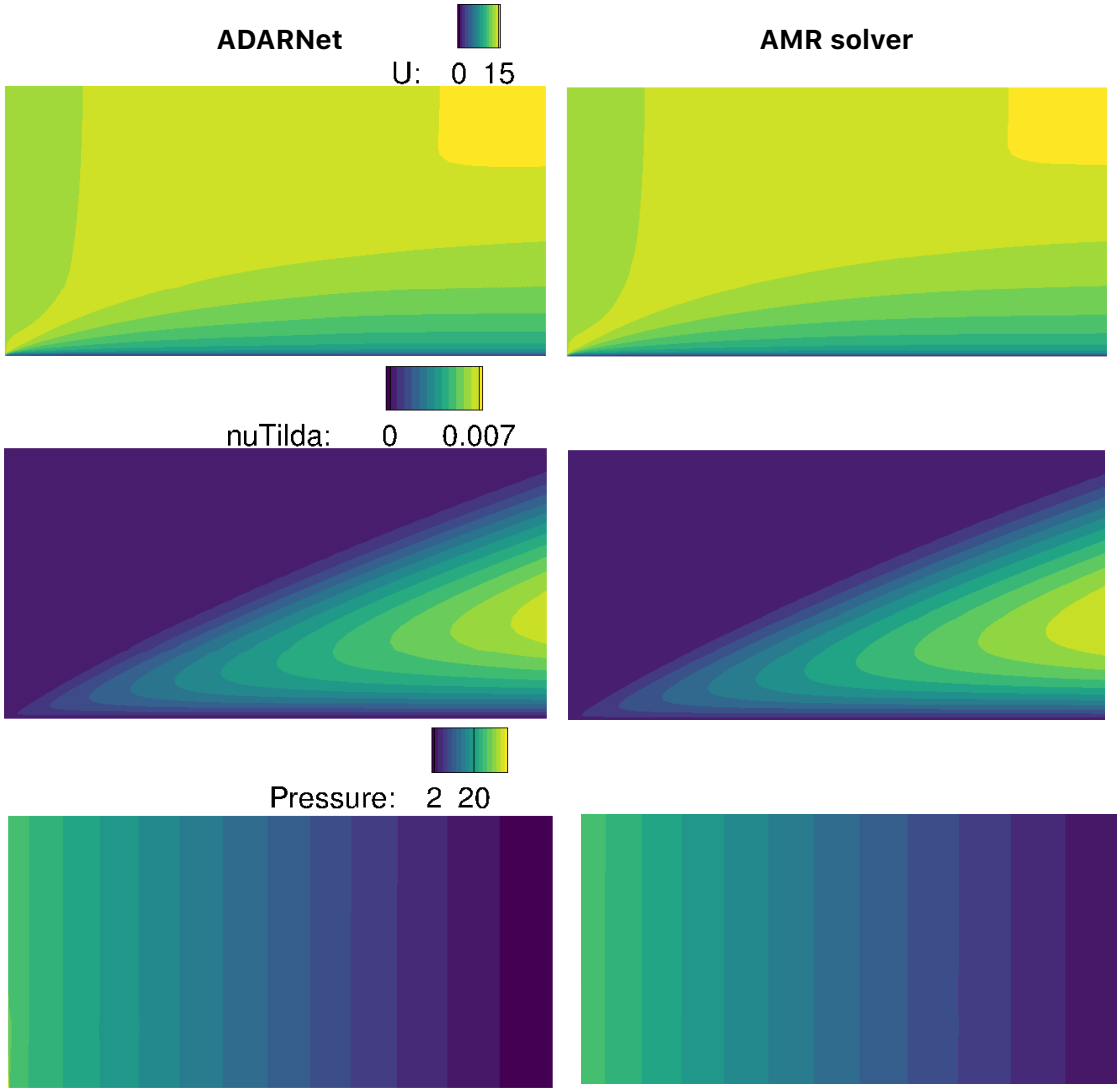


Figure 5.11: Velocity in $\mathrm{m\,s^{-1}}$ (top), kinematic pressure in $\mathrm{m^2/s^2}$ (bottom), and modified eddy viscosity in $\mathrm{m^2/s}$ (middle) for flat plate at $Re = 1.35 \times 10^6$. Comparison between ADARNet's result (left) and the AMR solver result (right) for $b = 4$ levels of refinement.

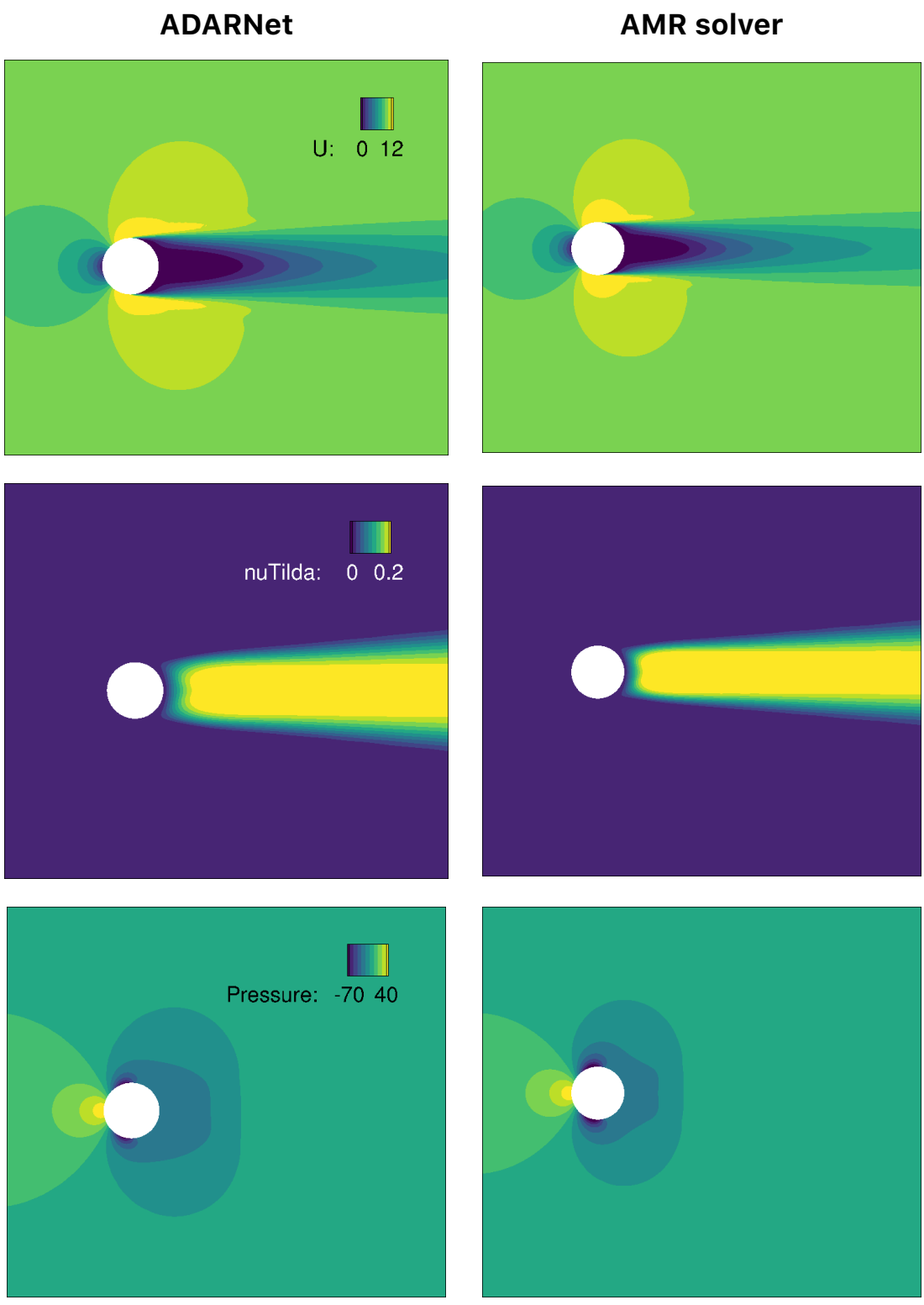


Figure 5.12: Velocity in $\mathrm{m\,s^{-1}}$ (top), kinematic pressure in $\mathrm{m^2/s^2}$ (bottom), and modified eddy viscosity in $\mathrm{m^2/s}$ (middle) for flow around a cylinder at $Re = 1 \times 10^5$. Comparison between ADARNet's result and the AMR solver result for $b = 4$ levels of refinement.

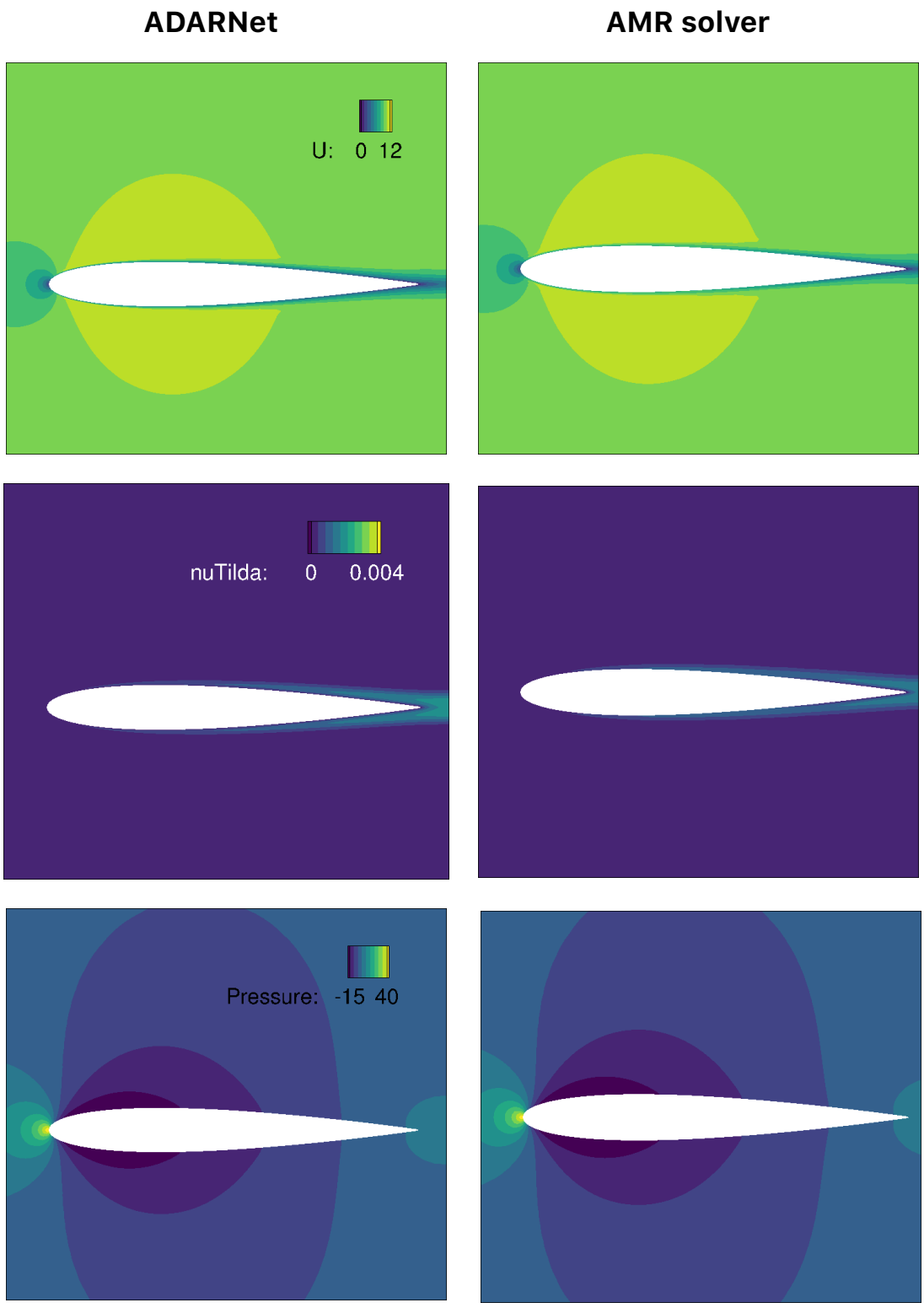


Figure 5.13: Velocity in $\mathrm{m\,s^{-1}}$ (top), kinematic pressure in $\mathrm{m^2/s^2}$ (bottom), and modified eddy viscosity in $\mathrm{m^2/s}$ (middle) for flow around a symmetric NACA0012 airfoil at $Re = 2.5 \times 10^4$. Comparison between ADARNet's result and the AMR solver result for $b = 4$ levels of refinement.

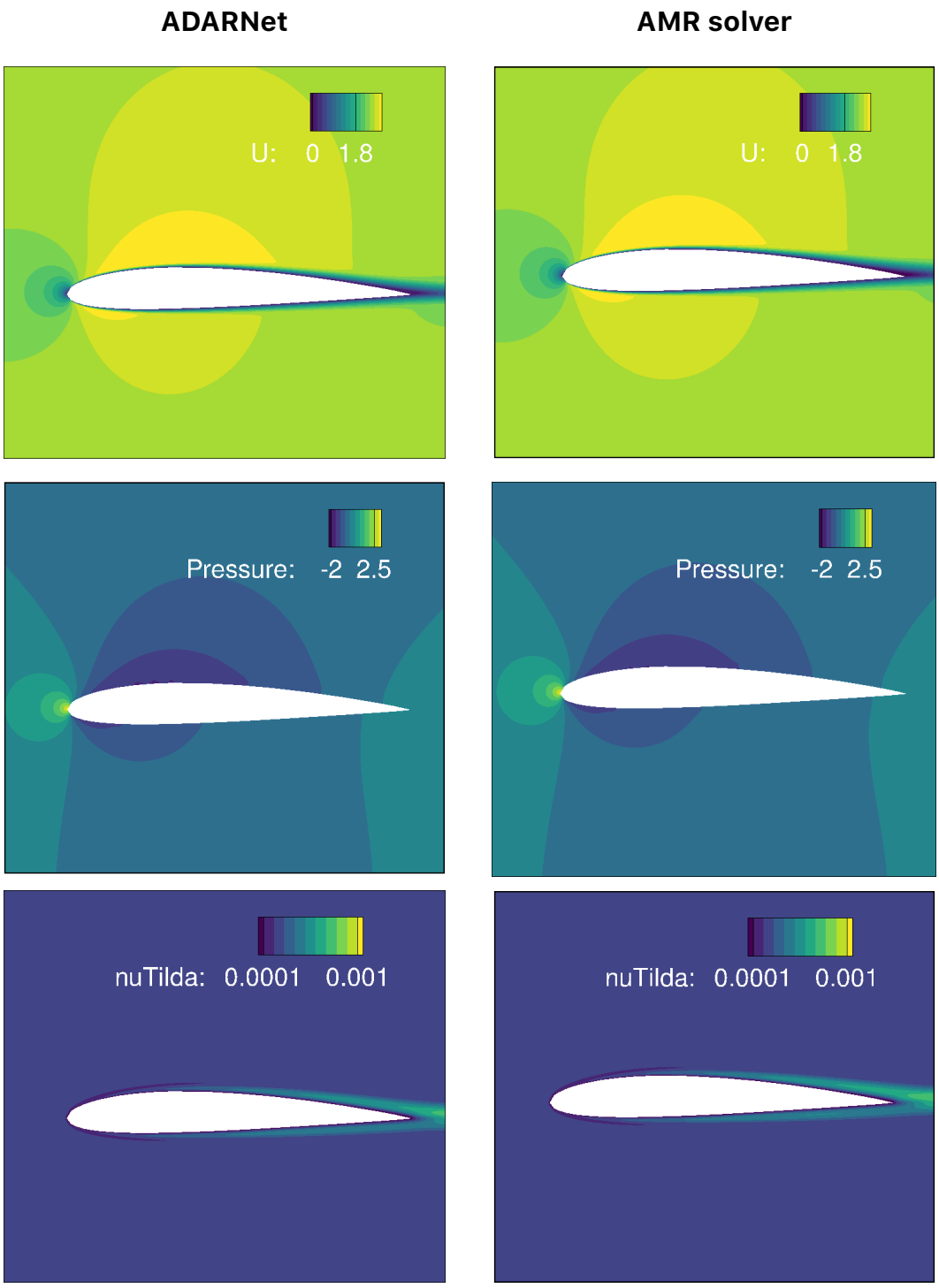


Figure 5.14: Velocity in $\mathrm{m\,s^{-1}}$ (top), kinematic pressure in $\mathrm{m^2/s^2}$ (bottom), and modified eddy viscosity in $\mathrm{m^2/s}$ (middle) for flow around a non-symmetric NACA1412 airfoil at $Re = 2.5 \times 10^4$. Comparison between ADARNet's result and the AMR solver result for $b = 4$ levels of refinement.

Quantitative results

We present a quantitative comparison between ADARNet and the AMR solver using a grid convergence study. We report, for the flat plate test cases, the coefficient of friction (C_f) at $x = 0.95L$, where L is the length of the flat plate. For the channel flow test cases, we also report the C_f on the lower wall at $x = 0.95L$. For the cylinder and airfoil test cases, we monitor the coefficient of drag or C_D. Figure 5.15 shows the value of the QoI for each test case with increasing refinement level n.

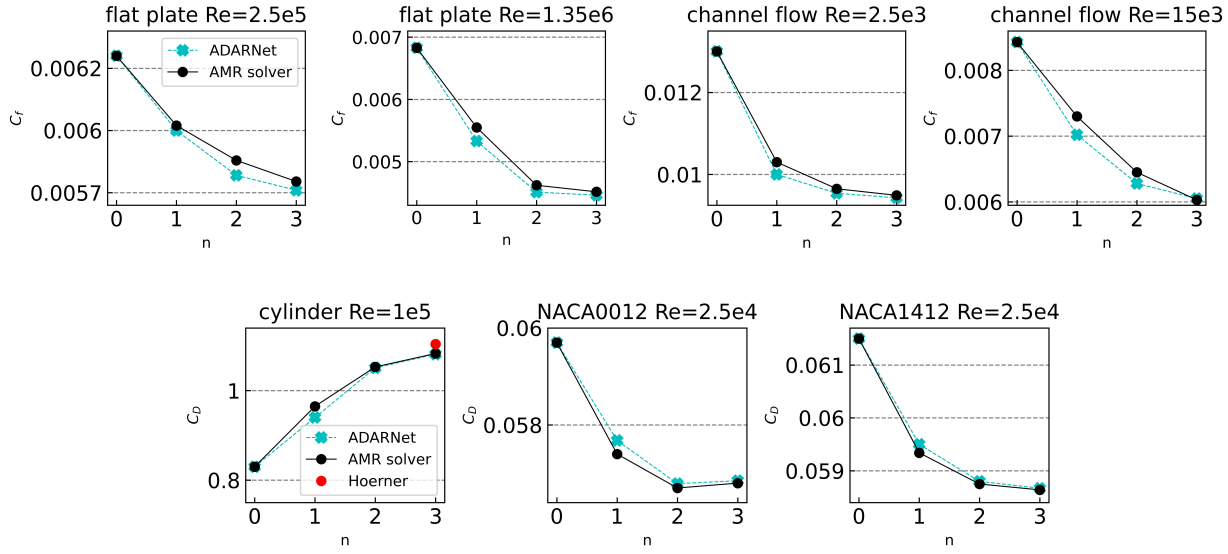


Figure 5.15: Value of the QoI versus refinement level n for ADARNet (blue) and the AMR solver (black) for each test case. C_f refers to coefficient of friction, and C_D to coefficient of drag. The red dot is the experimental value for the cylinder case found in [102]. Both algorithms converge as we increase the mesh refinement level from the original coarse mesh.

We make two main observations from the plots in Figure 5.15. First, we observe a good agreement between the QoI reported by ADARNet and the AMR solver at all levels of refinement. At $n = 0$, the value of the QoI is the same because they start with the same coarse mesh. Second, we observe how ADARNet's and the AMR solver's reported QoI show a notable convergence trend after $n = 1$. The plot referring to the cylinder case in Figure 5.15

shows, in red, the experimental value of C_D reported in Hoerner [102], which is 1.108, while ADARNet reports 1.0835, a 2.2% deviation, and the AMR solver reports 1.085, a 2.1% deviation. These errors are in line with those in the literature when comparing experimental results with RANS simulations using the SA model [103].

5.3.2 Performance analysis of ADARNet

In this section, we evaluate ADARNet's performance. We first compare its time-to-convergence (TTC) with the AMR solver's TTC in obtaining the results in Figure 5.15 at $n = 3$. Recall that ADARNet's TTC is the sum of the inference time and the time the physics solver takes to drive the solution from inference to convergence. Table 5.1 shows these times and reports the iterations-to-convergence (ITC) taken by both the physics solver and the AMR solver. For the channel flow case, ADARNet achieves a 4.3× speedup for the interpolated boundary condition case. For the flat plate, we obtain a 5.5× (interpolated case) and a 4.7× speedup (extrapolated case) over the AMR solver. ADARNet obtains an impressive 3.2× speedup for a flow around a cylinder, which is an unseen-during-training geometry. The cylinder case is the most challenging test case for ADARNet since accurately predicting the wake region behind the cylinder (as seen in Figure 5.12), a region with highly nonlinear, complex flow behavior, is challenging. Therefore, the physics solver spends a significant amount of time fixing that region and its speedup is the lowest among all test cases. Overall, ADARNet refines regions of interest such as near-wall areas (channel flow and flat plate) and the wake behind the solid body (cylinder), shows excellent grid convergence properties, and significantly accelerates the traditional AMR solver by $3.2 - 5.5\times$.

Next, we evaluate ADARNet's performance by comparing it with a baseline NN and present our results in Table 5.2. Recall that one of the goals of this thesis is to perform non-uniform super-resolution to avoid high-resolution inference in areas that do not require it.

	AMR solver		ADARNet		
	ITC	TTC	ITC	TTC inf + ps	speedup
channel flow $Re = 2.5 \times 10^3$	3369	3	2261	$1.2 \times 10^{-2} + 0.7$	4.3×
channel flow $Re = 15 \times 10^3$	4940	3.1	2022	$1.1 \times 10^{-1} + 0.8$	3.8×
flat plate $Re = 2.5 \times 10^5$	3389	2.7	1364	$8 \times 10^{-3} + 0.5$	5.5×
flat plate $Re = 1.35 \times 10^6$	5000	2	2214	$9 \times 10^{-3} + 0.4$	4.7×
cylinder $Re = 1 \times 10^5$	11155	4.8	4598	$6.3 \times 10^{-3} + 1.5$	3.2×
N0012 $Re = 2.5 \times 10^4$	2267	2	1150	$5 \times 10^{-3} + 0.6$	3.5×
N1412 $Re = 2.5 \times 10^4$	2637	2.1	1720	$5 \times 10^{-3} + 0.6$	3.3×

Table 5.1: Comparison of the time-to-convergence in minutes (TTC) and iterations-to-convergence (ITC) of ADARNet and the AMR solver. For ADARNet, we report separately the time spent in inference (inf) and the time spent by the physics solver (ps) driving the solution from inference to convergence, together with the speedup over the AMR solver.

We hypothesize that ADARNet is advantageous over SOTA methods that perform uniform super-resolution [53, 77, 90, 93] because these methods require 64× larger labels for 64× super-resolutions. To test our hypothesis, we build the SURFNet [93] framework and use it as our baseline. We compare ADARNet with SURFNet using two metrics. For a 64× super-resolution, we report, first, the time to achieve the same accuracy. The SURFNet's framework also consists of a DNN inference followed by a physics solver that guarantees convergence requirements. Hence, we compare both end-to-end frameworks and report both the inference time and the physics solver time. Second, we compare the memory consumption at inference. Because both ADARNet and SURFNet perform inference on a CPU, we report these metrics on the CPU described in Section 5.2.3.

Table 5.2 shows that ADARNet significantly outperforms SURFNet for a 64× super-resolution for all test cases. Specifically, we observe $7 - 28.5\times$ speedups over SURFNet. We observe the same behavior with the memory usage at inference. SURFNet requires almost 4 GB whereas ADARNet significantly reduces the memory consumption, realizing $4.4 - 7.5\times$ reduction factors. Note that ADARNet's inference time and memory usage is not consistent through the test cases because the fine/coarse regions change, as opposed to SURFNet that

	Memory usage			Time inf + ps		
	SURFNet	ADARNet	rf	SURFNet	ADARNet	speedup
cf $Re = 2.5 \times 10^3$	3.9	0.88	4.4×	0.25 + 14	$1.2 \times 10^{-2} + 0.7$	20.6×
cf $Re = 15 \times 10^3$	3.9	0.82	4.8×	0.25 + 14.5	$1.1 \times 10^{-2} + 0.8$	18.2×
fp $Re = 2.5 \times 10^5$	3.9	0.62	6.3×	0.25 + 11	$8 \times 10^{-3} + 0.5$	23×
fp $Re = 1.35 \times 10^6$	3.9	0.68	5.7×	0.25 + 12	$9 \times 10^{-3} + 0.4$	28.5×
cyl $Re = 1 \times 10^5$	3.9	0.52	7.5×	0.25 + 10.2	$6.3 \times 10^{-3} + 1.5$	7×
N0012 $Re = 2.5 \times 10^4$	3.9	0.54	7.2×	0.25 + 8.4	$5 \times 10^{-3} + 0.6$	15.5×
N1412 $Re = 2.5 \times 10^4$	3.9	0.51	7.7×	0.25 + 8.6	$5 \times 10^{-3} + 0.6$	14.1×

Table 5.2: Comparison of ADARNet with SURFNet. Left column compares the GB of memory consumed at each test case's inference and shows the reduction factor (rf) achieved by ADARNet. Right column compares, in minutes, the inference time (inf) and the time to convergence by the physics solver (ps) of both approaches and shows ADARNet's speedup over SURFNet. cf = channel flow, fp = flat plate, cyl = cylinder, N0012 = NACA0012 (symmetric airfoil), and N1412 = NACA1412 (non-symmetric airfoil).

performs uniform super-resolution.

5.4 Related work

AMR. AMR is a popular technique that makes it feasible to solve problems that are intractable on uniform grids and it has been widely applied in traditional finite volume-based solvers. When PDEs are solved numerically, they are often limited to a pre-determined computational grid or mesh. However, different areas of the domain can require different precisions where non-uniform grids are better suited. AMR algorithms adaptively and dynamically identify regions that require finer resolution (such as discontinuities, steep gradients, shocks, etc.) and refine or coarsen the mesh to achieve the target accuracy. Therefore, AMR can scale to resolutions that would otherwise be infeasible on uniform meshes resulting in increased computational efficiency and storage savings. Moreover, the adaptive strategy offers more control over the grid resolution compared to the fixed resolution of the static grid approaches.

The most popular AMR techniques apply to traditional finite volume/finite element based numerical solvers. Even though recent approaches have notably pushed the scalability boundaries of these systems [55, 94, 104] their core strategy results from the early work of Berger and Oliger [98], who introduced *local adaptive mesh refinement*. This algorithm starts with a coarse mesh from which certain cells are marked for refinement according to either a user-supplied criterion or based on the Richardson extrapolation [105]. The principle of marking cells for refinement is widespread. Two main approaches exist for identifying cells for refinement. First, adjoint-based AMR [106], which estimates the discretization error in each cell and adapts the mesh for lowering these errors. However, the optimal rationale for error estimation remains unknown [107]. Second, feature-based AMR [99], where the user supplies the variables (or features) to track and refines the computational cells that meet a user-defined value of those variables. Feature-based AMR is the most popular approach due to less challenging implementations and accurate results in a wide range of problems. However, feature-based AMR approaches require both a high degree of user intervention and expert, domain, and even specific knowledge of the problem at hand, and therefore have poor generalization properties. Existing AMR methods are based on a handful of heuristics whose long-term or general optimality remains unknown. To overcome this limitation, Yang et al. [108] designed the AMR procedure as a Markov Decision Process. However, the training is done with ground truth data generated from analytical solutions and can not be extended to turbulent flows.

There have been recent attempts to perform DL-based AMR. In [109], the authors increase the number of solution points in those areas where the residual is highest. However, this mesh-free method imposes the same refinement heuristics as traditional physical solvers and hence suffers from the same limitations. In [110], the authors develop AMRNet, a CNN-based model that performs *multi-resolution*, where the network outputs a uniform flow field at different resolutions. Since the output is uniform there is no discrimination between different areas of the flow. As opposed to the above approaches, we design a DL algorithm for AMR,

where the output is non-uniform and we do not impose any heuristic during the optimization process of the network. Instead, the training is guided by the governing equations of the problem that inform refining decisions

Super-resolution. DL algorithms have shown impressive results for super-resolution. We find super-resolution techniques applied to both CV and CFD problems. Two main research directions exist in CV: single-image super-resolution (SISR) and reference-based super-resolution (RefSR). However, both SISR and RefSR have a target resolution that is both known a priori and uniform [111–113]. In [113], the authors present the *texture transformer*, where the query, key, and value of their attention module are formed by upsampled and downsampled images of the input image together with a reference image from which textures are extracted. In [95], the authors provide a differentiable module that selects the most salient patches of the input image for image classification. However, the unselected patches are unused. In this thesis, we are interested in super-resolution, and therefore we keep all patches that cover the entire domain.

In CFD, we also find successful super-resolution attempts. Recent works use CNNs as finite-dimensional maps [77, 90, 91]. However, these approaches know the target resolution a priori, perform uniform super-resolution, and require large amounts of high-resolution labels. We presented SURFNet in Chapter 4 to eliminate the need of extensive data collection at high resolutions. However, SURFNet is also limited to uniform super-resolution. Mesh-free, resolution-invariant methods [50, 51, 53, 92] are a potential alternative to finite-dimensional maps because they can query the solution at any point in the domain and hence are prone to perform non-uniform super-resolution. Nevertheless, these methods do not intrinsically discriminate between different regions of the flow and hence end up yielding uniform output resolutions. Moreover, they also suffer from the limitation of extensive high-resolution data collection.

In this thesis, we present a semi-supervised DL algorithm that adaptively refines the input

mesh and outputs a non-uniform high-resolution flow field, improving both inference times and memory requirements for scaling to large problem sizes. ADARNet does not require knowledge of the target resolution a priori, hence eliminates both the need of collecting extensive high-resolution training data and the dependence on existing datasets that are limited to specific resolutions.

5.5 Conclusions of ADARNet

We presented ADARNet, a DL algorithm that predicts AMR. ADARNet is an end-to-end framework for non-uniform super-resolution of turbulent flows that predicts high-resolution accuracy only in specific regions of the domain while keeping areas with less complex flow features in the low-precision range for scalability and performance. ADARNet is trained with low-resolution data from three different canonical flows and predicts non-uniform flow fields for flow cases that have boundary conditions/geometries unseen during training. ADARNet shows excellent discerning properties in all test cases, producing higher resolution outputs in regions with complex flow features, such as near-wall areas or the wake region behind a cylinder, and keeping low-resolution patches in those areas that have smooth variations, such as the flow freestream.

ADARNet reaches the same convergence guarantees as traditional AMR solvers, shows excellent quantitative agreement with their heuristics (defined in Section 5.2.3), and accelerates them by $3.2 - 5.5\times$ in all test cases. Due to its ability to super-resolve only regions of interest, it reduces the end-to-end time and the memory usage by $7 - 28.5\times$ and $4.4 - 7.7\times$, respectively, over SOTA DL methods that perform uniform super-resolution.

The dynamic allocation of patches during training enables non-uniform super-resolution without any imposed and non-generalizable refinement heuristic. However, this impacts the

time-per-epoch, which increases significantly with respect to CFDNet and SURFNet.

Chapter 6

Concluding remarks

6.1 Summary

This dissertation has presented three novel hybrid DL-CFD frameworks for the acceleration of CFD simulations. We have shown that coupling a RANS solver with DNNs can significantly accelerate the convergence of the overall scheme.

First, 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, for small grid sizes, without relaxing the convergence constraints of the physics solver. To evaluate the framework's capacity for generalization and extrapolation, we tested CFDNet across various scenarios and geometries, including channel flow, ellipses, airfoils, and cylinders. In general, the framework performs well and demonstrates a capacity to make accurate predictions even for geometries unseen during training. CFDNet could be successfully applied, for instance, to Large Eddy Simulations (LES). In principle, any discretized field of inputs should be amenable to CFDNet. Moreover, a wide variety of domains in scientific computing, such as molecular dynamics and material science, could be considered.

Second, to scale CFDNet to large-grid simulations, we presented SURFNet, a super-resolution flow network that accelerates high-resolution turbulent CFD simulations. SURFNet is primarily trained on low-resolution simulation data and applies this information (via transfer learning) to high-resolution inputs. We presented two variations, one-shot transfer learning (OSTL) and incremental transfer learning (ITL). Both approaches yield consistent speedups across test geometries unseen during the training or transfer stages and exhibit good generalization capacities. We demonstrated resolution-invariance with ITL on domains up to $256\times$ larger than the tiny discretization used in training and a uniform $2\times$ speedup across target resolutions and test geometries compared to the OpenFOAM CFD solver. SURFNet can recover high-resolution flow features with $15\times$ fewer data at high resolutions and reduce the combined data collection and training time by $3.6\times$ and $10.2\times$ at 256×256 and 512×512 grid sizes, respectively.

To overcome SURFNet's limitation of uniform super-resolution and reach high-resolution inferences with less computational resources, we presented ADARNet, a DL-based framework for AMR. ADARNet is an end-to-end framework for non-uniform super-resolution of turbulent flows. ADARNet predicts high-resolution accuracy only in specific domain regions while keeping areas with smoother complex flow features in the low-precision range for scalability and performance. ADARNet reaches the same convergence guarantees as traditional AMR solvers, shows excellent quantitative agreement with their heuristics, and accelerates them by $3.2 - 5.5\times$ in all test cases. Due to its ability to super-resolve only regions of interest, it reduces the end-to-end time and the memory usage by $7-28.5\times$ and $4.4-7.7\times$, respectively, over SOTA DL methods that perform uniform super-resolution.

6.2 Future directions

6.2.1 Improving the accuracy of DL for CFD

CFDNet, SURFNet, and ADARNet demonstrate that the training and validation losses are far from reaching the accuracy of traditional finite volume-based solvers. We also observe this trend in this dissertation's related work. This gap of several orders of magnitude is due to the complexity of the loss function and the inability of current gradient descent algorithms to reach better local minima. This is incredibly complex when adding not one, but several second-order, non-linear PDEs in the loss function. Current DL algorithms need to improve to become valid surrogates of expensive finite volume solvers. A potential research direction is to perform the optimization of the PDE residual in its Fourier or Laplace space, which can smooth out the landscape of the loss function and help reach better local optima. Moreover, it is also worth studying more optimizations in the network, such as new batch/layer normalization layers that can help reach lower training and validation losses.

6.2.2 A DL algorithm for multigrid

This thesis has shown that coupling traditional methods with DL is a promising approach. A natural research direction is to find a domain-agnostic approximator for multigrid (MG). MG algorithms successfully accelerate CFD simulations because they reduce the required iterations-to-convergence. However, MG algorithms have a fundamental limitation: each iteration is more expensive than the non-MG iteration. A promising research direction is to couple the best of both worlds: fast DNN inferences and a reduced number of iterations from MG. MG is an iterative method itself, and hence expensive. We propose a DL model to replace the iterative nature of the MG algorithm and provide the next iteration of the physical system in one step. A hybrid framework (for instance, the solver returns to the

traditional MG algorithm if too much error is accumulated) could also be explored. This model would deliver fewer iterations-to-convergence and reduce the time per iteration while being physically consistent. As a result, we would significantly speedup the overall scheme and generalize to new problems for which the DNN does not require prior knowledge.

Bibliography

[1] S. B. Pope, "Turbulent flows," 2001.

[2] O. Reynolds, "Iv. on the dynamical theory of incompressible viscous fluids and the determination of the criterion," *Philosophical transactions of the royal society of london.(a.)*, no. 186, pp. 123–164, 1895.

[3] P. Yeung, X. Zhai, and K. R. Sreenivasan, "Extreme events in computational turbulence," *Proceedings of the National Academy of Sciences*, vol. 112, no. 41, pp. 12 633–12 638, 2015.

[4] J. Slotnick, A. Khodadoust, J. Alonso, D. Darmofal, W. Gropp, E. Lurie, and D. Mavriplis, "Cfd vision 2030 study: a path to revolutionary computational aerosciences," 2014.

[5] J. Ling, A. Kurzawski, and J. Templeton, "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance," *Journal of Fluid Mechanics*, vol. 807, pp. 155–166, 2016.

[6] A. Van den Oord, S. Dieleman, and B. Schrauwen, "Deep content-based music recommendation," in *Advances in neural information processing systems*, 2013, pp. 2643–2651.

[7] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 160–167.

[8] U. Ghia, K. N. Ghia, and C. Shin, "High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method," *Journal of computational physics*, vol. 48, no. 3, pp. 387–411, 1982.

[9] M. Lee, N. Malaya, and R. D. Moser, "Petascale direct numerical simulation of turbulent channel flow on up to 786k cores," pp. 1–11, 2013.

[10] F. Salvadore, M. Bernardini, and M. Botti, "Gpu accelerated flow solver for direct numerical simulation of turbulent flows," *Journal of Computational Physics*, vol. 235, pp. 129–142, 2013.

[11] B. Mostafazadeh, F. Marti, F. Liu, and A. Chandramowlishwaran, "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, May 2018, pp. 753–762.

[12] Z. Lilek, S. Muzaferija, and M. Perić, "Efficiency and accuracy aspects of a full-multigrid simple algorithm for three-dimensional flows," *Numerical Heat Transfer*, vol. 31, no. 1, pp. 23–42, 1997.

[13] S. Deng, L. Jiang, and C. Liu, "Dns for flow separation control around an airfoil by pulsed jets," *Computers & fluids*, vol. 36, no. 6, pp. 1040–1060, 2007.

[14] P. Spalart and S. Allmaras, "A one-equation turbulence model for aerodynamic flows," in *30th aerospace sciences meeting and exhibit*, 1992, p. 439.

[15] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–43, 2019.

[16] E. Monmasson, L. Idkhajine, M. N. Cirstea, I. Bahri, A. Tisan, and M. W. Naouar, "Fpgas in industrial control applications," *IEEE Transactions on Industrial informatics*, vol. 7, no. 2, pp. 224–243, 2011.

[17] S. L. Krist, *CFL3D user's manual (version 5.0)*. National Aeronautics and Space Administration, Langley Research Center, 1998.

[18] D. Jespersen, T. Pulliam, P. Buning, D. Jespersen, T. Pulliam, and P. Buning, "Recent enhancements to overflow," in *35th Aerospace Sciences Meeting and Exhibit*, 1997, p. 644.

[19] R. T. Biedron, J. R. Carlson, J. M. Derlaga, P. A. Gnoffo, D. P. Hammond, W. T. Jones, B. Kleb, E. M. Lee-Rausch, E. J. Nielsen, M. A. Park *et al.*, "Fun3d manual: 13.6," 2019.

[20] H. Jasak, A. Jemcov, Z. Tukovic *et al.*, "Openfoam: A c++ library for complex physics simulations," in *International workshop on coupled methods in numerical dynamics*, vol. 1000. IUC Dubrovnik Croatia, 2007, pp. 1–20.

[21] T. Akiba, S. Suzuki, and K. Fukuda, "Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes," *arXiv preprint arXiv:1711.04325*, 2017.

[22] J. Hensman, N. Fusi, and N. D. Lawrence, "Gaussian processes for big data," *arXiv preprint arXiv:1309.6835*, 2013.

[23] H. Fang, M. Rais-Rohani, Z. Liu, and M. Horstemeyer, "A comparative study of metamodeling methods for multiobjective crashworthiness optimization," *Computers & Structures*, vol. 83, no. 25-26, pp. 2121–2136, 2005.

[24] M. J. Garcia, P. Boulanger, and S. Giraldo, "Cfd based wing shape optimization through gradient-based method," in *Proceedings of the International Conference on Engineering Optimization. EngOpt, Rio de Janeiro, Brazil*, 2008.

[25] S. Koziel and L. Leifsson, "Multi-level cfd-based airfoil shape optimization with automated low-fidelity model selection," *Procedia Computer Science*, vol. 18, pp. 889–898, 2013.

[26] L. Huyse, S. L. Padula, R. M. Lewis, and W. Li, "Probabilistic approach to free-form airfoil shape optimization under uncertainty," *AIAA journal*, vol. 40, no. 9, pp. 1764–1772, 2002.

[27] M. Khurana, H. Winarto, and A. Sinha, "Airfoil geometry parameterization through shape optimizer and computational fluid dynamics," in *46th AIAA Aerospace Sciences Meeting and Exhibit*, 2008, p. 295.

[28] "NACA0012," https://turbmodels.larc.nasa.gov/naca0012_val.html.

[29] D. W. Levy, T. Zickuhr, J. Vassberg, S. Agrawal, R. A. Wahls, S. Pirzadeh, and M. J. Hemsch, "Data summary from the first aiaa computational fluid dynamics drag prediction workshop," *Journal of Aircraft*, vol. 40, no. 5, pp. 875–882, 2003.

[30] B. Mostafazadeh, F. Marti, F. Liu, and A. Chandramowlishwaran, "Roofline guided design and analysis of a multi-stencil cfd solver for multicore performance," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018, pp. 753–762.

[31] S. Blagodurov, A. Fedorova, S. Zhuravlev, and A. Kamali, "A case for numa-aware contention management on multicore systems," in *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2010, pp. 557–558.

[32] H. Wang and A. Chandramowlishwaran, "Multi-criteria partitioning of multi-block structured grids," in *Proceedings of the ACM International Conference on Supercomputing*, 2019, pp. 261–271.

[33] A. A. Giunta, "Aircraft multidisciplinary design optimization using design of experiments theory and response surface modeling methods," Ph.D. dissertation, Virginia Tech, 1997.

[34] V. O. Balabanov, A. A. Giunta, O. Golovidov, B. Grossman, W. H. Mason, L. T. Watson, and R. T. Haftka, "Reasonable design space approach to response surface approximation," *Journal of Aircraft*, vol. 36, no. 1, pp. 308–315, 1999.

[35] A. Pollard, T. J. Hacker, and S. Dyke, "Whither turbulence and big data for the twenty-first century," pp. 551–574, 2017.

[36] E. Perlman, R. Burns, Y. Li, and C. Meneveau, "Data exploration of turbulence simulations using a database cluster," in *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, 2007, pp. 1–11.

[37] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[38] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[39] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin, "Accelerating eulerian fluid simulation with convolutional networks," pp. 3424–3433, 2017.

[40] X. Guo, W. Li, and F. Iorio, "Convolutional neural networks for steady flow approximation," pp. 481–490, 2016.

[41] R. Maulik, H. Sharma, S. Patel, B. Lusch, and E. Jennings, "Accelerating rans turbulence modeling using potential flow and machine learning," *arXiv preprint arXiv:1910.10878*, 2019.

[42] N. Thuerey, K. Weißenow, L. Prantl, and X. Hu, "Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows," *AIAA Journal*, pp. 1–12, 2019.

[43] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational physics*, vol. 378, pp. 686–707, 2019.

[44] E. Haghighat and R. Juanes, "Sciann: A keras/tensorflow wrapper for scientific computations and physics-informed deep learning using artificial neural networks," *Computer Methods in Applied Mechanics and Engineering*, vol. 373, p. 113552, 2021.

[45] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "Deepxde: A deep learning library for solving differential equations," *SIAM Review*, vol. 63, no. 1, pp. 208–228, 2021.

[46] O. Hennigh, S. Narasimhan, M. A. Nabian, A. Subramaniam, K. Tangsali, Z. Fang, M. Rietmann, W. Byeon, and S. Choudhry, "Nvidia simnet$^{\text{TM}}$: An ai-accelerated multi-physics simulation framework," in *International Conference on Computational Science*. Springer, 2021, pp. 447–461.

[47] S. Wang, Y. Teng, and P. Perdikaris, "Understanding and mitigating gradient pathologies in physics-informed neural networks," *arXiv preprint arXiv:2001.04536*, 2020.

[48] H. Wang, R. Planas, A. Chandramowlishwaran, and R. Bostanabad, "Mosaic flows: A transferable deep learning framework for solving pdes on unseen domains," *Computer Methods in Applied Mechanics and Engineering*, vol. 389, p. 114424, 2022.

[49] L. Lu, P. Jin, and G. E. Karniadakis, "Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators," *arXiv preprint arXiv:1910.03193*, 2019.

[50] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations," *arXiv preprint arXiv:2010.08895*, 2020.

[51] ——, "Neural operator: Graph kernel network for partial differential equations," *arXiv preprint arXiv:2003.03485*, 2020.

[52] K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart, "Model reduction and neural networks for parametric pdes," *arXiv preprint arXiv:2005.03180*, 2020.

[53] C. M. Jiang, S. Esmaeilzadeh, K. Azizzadenesheli, K. Kashinath, M. Mustafa, H. A. Tchelepi, P. Marcus, A. Anandkumar *et al.*, "Meshfreeflownet: A physics-constrained deep continuous space-time super-resolution framework," *arXiv preprint arXiv:2005.01463*, 2020.

[54] W. Dong, J. Liu, Z. Xie, and D. Li, "Adaptive neural network-based approximation to accelerate eulerian fluid simulation," p. 7, 2019.

[55] W. Zhang, A. Myers, K. Gott, A. Almgren, and J. Bell, "AMReX: Block-structured adaptive mesh refinement for multiphysics applications," *The International Journal of High Performance Computing Applications*, p. 10943420211022811, 2021.

[56] J. Boussinesq, *Essai sur la théorie des eaux courantes*. Impr. nationale, 1877.

[57] M. F. Møller, *A scaled conjugate gradient algorithm for fast supervised learning*. Aarhus University, Computer Science Department, 1990.

[58] H. B. Barlow, "Unsupervised learning," *Neural computation*, vol. 1, no. 3, pp. 295–311, 1989.

[59] S. Patankar, *Numerical heat transfer and fluid flow*. CRC press, 2018.

[60] A. Okajima, D. Yi, A. Sakuda, and T. Nakano, "Numerical study of blockage effects on aerodynamic characteristics of an oscillating rectangular cylinder," *Journal of wind engineering and industrial aerodynamics*, vol. 67, pp. 91–102, 1997.

[61] S. Muzaferija and D. Gosman, "Finite-volume cfd procedure and adaptive error control strategy for grids of arbitrary topology," *Journal of computational physics*, vol. 138, no. 2, pp. 766–787, 1997.

[62] J. R. Cash and A. H. Karp, "A variable order runge-kutta method for initial value problems with rapidly varying right-hand sides," *ACM Transactions on Mathematical Software (TOMS)*, vol. 16, no. 3, pp. 201–222, 1990.

[63] Y. Liu, E. Racah, Prabhat, J. Correa, A. Khosrowshahi, D. Lavers, K. Kunkel, M. Wehner, and W. Collins, "Application of deep convolutional neural networks for detecting extreme weather in climate datasets," *arXiv preprint arXiv:1605.01156*, 2016.

[64] L. Zhu, W. Zhang, J. Kou, and Y. Liu, "Machine learning methods for turbulence modeling in subsonic flows around airfoils," *Physics of Fluids*, vol. 31, no. 1, p. 015105, 2019.

[65] C. K. Batchelor and G. Batchelor, *An introduction to fluid dynamics*. Cambridge university press, 2000.

[66] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," pp. 1026–1034, 2015.

[67] M. Mustafa, D. Bard, W. Bhimji, Z. Lukić, R. Al-Rfou, and J. M. Kratochvil, "Cosmogan: creating high-fidelity weak lensing convergence maps using generative adversarial networks," *Computational Astrophysics and Cosmology*, vol. 6, no. 1, pp. 1–13, 2019.

[68] N. B. Erichson, M. Muehlebach, and M. W. Mahoney, "Physics-informed autoencoders for lyapunov-stable fluid flow prediction," *arXiv preprint arXiv:1905.10866*, 2019.

[69] M. Ahmed and N. Qin, "Surrogate-based aerodynamic design optimization: use of surrogates in aerodynamic design optimization," in *International Conference on Aerospace Sciences and Aviation Technology*, vol. 13, no. AEROSPACE SCIENCES & AVIATION TECHNOLOGY, ASAT-13, May 26–28, 2009. The Military Technical College, 2009, pp. 1–26.

[70] "NACA0012 grids," https://turbmodels.larc.nasa.gov/naca0012_grids.html.

[71] A. Gulli and S. Pal, *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[72] H. Shan, L. Jiang, and C. Liu, "Direct numerical simulation of flow separation around a naca 0012 airfoil," *Computers & fluids*, vol. 34, no. 9, pp. 1096–1114, 2005.

[73] National Advisory Committee for Aeronautics airfoils, "NACA Family of Airfoils," https://www.nasa.gov/image-feature/langley/100/naca-airfoils, 1929.

[74] E. Nehme, L. E. Weiss, T. Michaeli, and Y. Shechtman, "Deep-storm: super-resolution single-molecule microscopy by deep learning," *Optica*, vol. 5, no. 4, pp. 458–464, 2018.

[75] W. Yang, X. Zhang, Y. Tian, W. Wang, J.-H. Xue, and Q. Liao, "Deep learning for single image super-resolution: A brief review," *IEEE Transactions on Multimedia*, vol. 21, no. 12, pp. 3106–3121, 2019.

[76] Z. Wang, J. Chen, and S. C. Hoi, "Deep learning for image super-resolution: A survey," *IEEE transactions on pattern analysis and machine intelligence*, 2020.

[77] H. Gao, L. Sun, and J.-X. Wang, "Super-resolution and denoising of fluid flow using physics-informed convolutional neural networks without high-resolution labels," *arXiv preprint arXiv:2011.02364*, 2020.

[78] R. Wang, K. Kashinath, M. Mustafa, A. Albert, and R. Yu, "Towards physics-informed deep learning for turbulent flow prediction," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1457–1466.

[79] O. Obiols-Sales, A. Vishnu, N. Malaya, and A. Chandramowliswharan, "Cfdnet: A deep learning-based accelerator for fluid simulations," in *Proceedings of the 34th ACM International Conference on Supercomputing*, 2020, pp. 1–12.

[80] W. Peng, Y. Zhang, and M. Desmarais, "Spatial convolution neural network for efficient prediction of aerodynamic coefficients," in *AIAA Scitech 2021 Forum*, 2021, p. 0277.

[81] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *International conference on artificial neural networks*. Springer, 2018, pp. 270–279.

[82] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[83] C. Cai, S. Wang, Y. Xu, W. Zhang, K. Tang, Q. Ouyang, L. Lai, and J. Pei, "Transfer learning for drug discovery," *Journal of Medicinal Chemistry*, vol. 63, no. 16, pp. 8683–8694, 2020.

[84] Y. Pathak, P. K. Shukla, A. Tiwari, S. Stalin, and S. Singh, "Deep transfer learning based classification model for covid-19 disease," *Irbm*, 2020.

[85] S. Ruder, "Neural transfer learning for natural language processing," Ph.D. dissertation, NUI Galway, 2019.

[86] S. Shao, S. McAleer, R. Yan, and P. Baldi, "Highly accurate machine fault diagnosis using deep transfer learning," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2446–2455, 2018.

[87] J. Casacuberta, K. Groot, Q. Ye, and S. Hickel, "Transitional flow dynamics behind a micro-ramp," *Flow, Turbulence and Combustion*, vol. 104, no. 2, pp. 533–552, 2020.

[88] M. Long, H. Zhu, J. Wang, and M. I. Jordan, "Deep transfer learning with joint adaptation networks," in *International conference on machine learning*. PMLR, 2017, pp. 2208–2217.

[89] E. Ott, *Chaos in dynamical systems*. Cambridge university press, 2002.

[90] Fukami, Kai and Fukagata, Koji and Taira, Kunihiko, "Machine-learning-based spatio-temporal super resolution reconstruction of turbulent flows," *Journal of Fluid Mechanics*, vol. 909, 2021.

[91] B. Liu, J. Tang, H. Huang, and X.-Y. Lu, "Deep learning methods for super-resolution reconstruction of turbulent flows," *Physics of Fluids*, vol. 32, no. 2, p. 025105, 2020.

[92] K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart, "Model reduction and neural networks for parametric pdes," *arXiv preprint arXiv:2005.03180*, 2020.

[93] O. Obiols-Sales, A. Vishnu, N. P. Malaya, and A. Chandramowlishwaran, "Surfnet: Super-resolution of turbulent flows with transfer learning using small datasets," in *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2021, pp. 331–344.

[94] G. L. Bryan, M. L. Norman, B. W. O'Shea, T. Abel, J. H. Wise, M. J. Turk, D. R. Reynolds, D. C. Collins, P. Wang, S. W. Skillman *et al.*, "Enzo: An adaptive mesh refinement code for astrophysics," *The Astrophysical Journal Supplement Series*, vol. 211, no. 2, p. 19, 2014.

[95] J.-B. Cordonnier, A. Mahendran, A. Dosovitskiy, D. Weissenborn, J. Uszkoreit, and T. Unterthiner, "Differentiable patch selection for image recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 2351–2360.

[96] L. B. Rall and G. F. Corliss, "An introduction to automatic differentiation," *Computational Differentiation: Techniques, Applications, and Tools*, vol. 89, 1996.

[97] P. Moin and J. Kim, "Numerical investigation of turbulent channel flow," *Journal of fluid mechanics*, vol. 118, pp. 341–377, 1982.

[98] M. J. Berger and J. Oliger, "Adaptive mesh refinement for hyperbolic partial differential equations," *Journal of computational Physics*, vol. 53, no. 3, pp. 484–512, 1984.

[99] N. Kasmai, D. Thompson, E. Luke, M. Jankun-Kelly, and R. Machiraju, "Feature-based adaptive mesh refinement for wingtip vortices," *International journal for numerical methods in fluids*, vol. 66, no. 10, pp. 1274–1294, 2011.

[100] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[101] L. Eça and M. Hoekstra, "A procedure for the estimation of the numerical uncertainty of cfd calculations based on grid refinement studies," *Journal of computational physics*, vol. 262, pp. 104–130, 2014.

[102] S. F. Hoerner, "Fluid-dynamic drag," *Hoerner fluid dynamics*, 1965.

[103] Y. Bao, D. Zhou, C. Huang, Q. Wu, and X.-q. Chen, "Numerical prediction of aerodynamic characteristics of prismatic cylinder by finite element method with spalart–allmaras turbulence model," *Computers & structures*, vol. 89, no. 3-4, pp. 325–338, 2011.

[104] A. Dubey, M. Berzins, C. Burstedde, M. L. Norman, D. Unat, and M. Wahib, "Structured adaptive mesh refinement adaptations to retain performance portability with increasing heterogeneity," *Computing in Science Engineering*, vol. 23, no. 5, pp. 62–66, 2021.

[105] "Richardson Extrapolation," https://personal.math.ubc.ca/~israel/m215/rich/rich.html.

[106] M. Nemec, M. Aftosmis, and M. Wintzer, "Adjoint-based adaptive mesh refinement for complex geometries," in *46th AIAA Aerospace Sciences Meeting and Exhibit*, 2008, p. 725.

[107] J. Bohn and M. Feischl, "Recurrent neural networks as optimal mesh refinement strategies," *Computers & Mathematics with Applications*, vol. 97, pp. 61–76, 2021.

[108] J. Yang, T. Dzanic, B. Petersen, J. Kudo, K. Mittal, V. Tomov, J.-S. Camier, T. Zhao, H. Zha, T. Kolev *et al.*, "Reinforcement learning for adaptive mesh refinement," *arXiv preprint arXiv:2103.01342*, 2021.

[109] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "Deepxde: A deep learning library for solving differential equations," *SIAM Review*, vol. 63, no. 1, pp. 208–228, 2021.

[110] Y. Asahi, S. Hatayama, T. Shimokawabe, N. Onodera, Y. Hasegawa, and Y. Idomura, "Amr-net: Convolutional neural networks for multi-resolution steady flow prediction," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2021, pp. 686–691.

[111] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *European conference on computer vision*. Springer, 2014, pp. 184–199.

[112] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4681–4690.

[113] F. Yang, H. Yang, J. Fu, H. Lu, and B. Guo, "Learning texture transformer network for image super-resolution," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 5791–5800.