ADARNet: Deep Learning Predicts Adaptive Mesh Refinement

Octavi Obiols-Sales
oobiols@uci.edu
University of California, Irvine
Irvine, California, USA

Abhinav Vishnu
abhinav.vishnu@amd.com
Advanced Micro Devices
Austin, Texas, USA

Nicholas Malaya
nicholas.malaya@amd.com
Advanced Micro Devices
Austin, Texas, USA

Aparna Chandramowlishwaran
amowli@uci.edu
University of California, Irvine
Irvine, California, USA

ABSTRACT

Deep Learning (DL) algorithms have gained popularity for super-resolution tasks - reconstructing a high-resolution (HR) output from its low-resolution (LR) counterpart. However, current DL approaches, both in computer vision and computational fluid dynamics (CFD), perform spatially uniform super-resolution. Therefore, DL for CFD approaches often over-resolve regions of the LR input that are already accurate at low numerical precision. This hardware over-utilization limits their scalability. To address this limitation, we propose ADARNet, a DL-based adaptive mesh refinement (AMR) framework. ADARNet takes a LR image as input and outputs its non-uniform HR counterpart, predicting HR only in areas that require higher numerical accuracy. As a result, ADARNet predicts the target 1024×1024 solution $7 - 28.5\times$ faster than state-of-the-art DL methods and reduces the memory usage by $4.4 - 7.65\times$ while maintaining the same level of accuracy. Moreover, unlike traditional AMR solvers that refine the mesh iteratively, ADARNet is a one-shot method that accelerates it by $2.6 - 4.5\times$.

CCS CONCEPTS

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

KEYWORDS

Physics-informed machine learning, adaptive mesh refinement, super-resolution, turbulent flows

ACM Reference Format:
Octavi Obiols-Sales, Abhinav Vishnu, Nicholas Malaya, and Aparna Chandramowlishwaran. 2023. ADARNet: Deep Learning Predicts Adaptive Mesh Refinement. In *52nd International Conference on Parallel Processing (ICPP 2023), August 07–10, 2023, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3605573.3605654

This work is licensed under a Creative Commons Attribution International 4.0 License.

ICPP 2023, August 07–10, 2023, Salt Lake City, UT, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0843-5/23/08.
https://doi.org/10.1145/3605573.3605654

1 INTRODUCTION

Computational Fluid Dynamics (CFD) is the *de-facto* method for solving the Navier-Stokes equations, which govern fluid flow behavior. However, CFD simulations often require high spatial resolutions to accurately capture complex flow phenomena, making them computationally expensive. There are widespread efforts to address this challenge and to improve the performance and scalability of solving these systems [7, 11, 20, 21, 23]. Inspired by the remarkable success of deep learning (DL) algorithms in various fields, including computer vision (CV) and natural language processing (NLP), recent efforts have explored using DL algorithms for accelerating CFD simulations via *super-resolution* (SR). SR involves reconstructing expensive high-resolution (HR) solutions from their low-resolution (LR) counterpart, which are computationally cheaper to simulate [9, 10, 14, 24].

The state-of-the-art (SOTA) DL models for SR have shown promise as real-time predictors [9, 14] that can generalize well across a wide range of flow configurations. However, these models share a common limitation: they perform *uniform* SR, that is, *every* pixel of the input LR image is refined to the target high-resolution output. As a result, uniform SR methods [9, 14, 24] impose higher computational demands, resulting in increased inference times and memory requirements. This limitation is particularly evident when the target spatial resolution is high, such as 1024×1024, which is necessary to accurately model flow phenomena like the boundary layer. Figure 1 shows the maximum allowable batch size with increasing target spatial resolution. The SOTA SR methods in CFD do not allow more than two samples per batch during inference on a 16GB NVIDIA V100 GPU at high spatial resolutions, significantly restricting their practical usability for accelerating design space exploration in CFD.

Spatially uniform outputs are computationally inefficient for two additional reasons. First, they tend to over-resolve regions with smooth fluctuations in the flow properties. In fluid dynamics, many flow phenomena exhibit smooth variations over certain regions, and applying high-resolution refinement uniformly across the entire domain may lead to excessive detail in these areas. This not only wastes computational resources but can also introduce noise into the results. Second, current uniform SR approaches require prior knowledge of the target resolution. This requirement necessitates a large number of HR labels at that specific resolution, making the data collection process computationally challenging and resource-intensive [9, 14, 19]. Moreover, obtaining a vast amount of HR data can be impractical, especially when dealing with large-scale simulations or complex flow configurations.

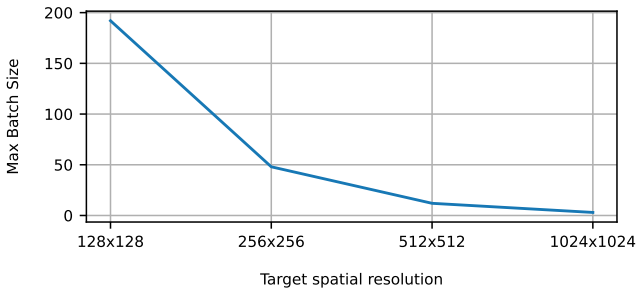

Figure 1: Maximum possible batch size during inference at different target spatial resolutions for SURFNet [24], a SOTA DL SR model on a 16GB NVIDIA V100 GPU.

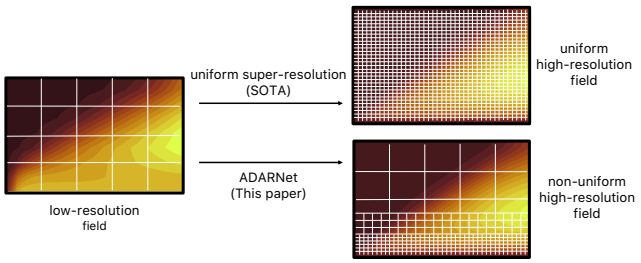

Figure 2: Current DL algorithms for SR output the solution on a uniform fine mesh (top). Our objective with ADARNet is to predict a spatially non-uniform output where only areas that require higher accuracy are refined (bottom). Hence, ADARNet requires less compute time and memory resources while achieving the target accuracy.

To address the above challenge, traditional numerical solvers often employ adaptive mesh refinement (AMR). AMR is a technique that selectively refines only those regions in the computational domain that exhibit significant flow variability, while leaving other regions coarser. This adaptive refinement helps achieve better scalability and performance in large-scale simulations [5, 30]. However, conventional AMR methods in CFD suffer from two main limitations. First, they heavily rely on user intervention and heuristics that require specific knowledge of the problem, making them less generalizable. Second, the mesh refinement process is iterative. It involves multiple iterations of solving the flow equations, assessing the solution, and then refining or coarsening the mesh based on certain criteria. This iterative nature demands more computational resources and time compared to direct methods.

In this paper, we tackle all these challenges and present ADARNet, a novel DL framework for **ADA**ptive mesh **R**efinement. ADARNet takes as input a LR flow field and outputs, in one-shot, its final non-uniform HR solution, as seen in Figure 2. Since only regions in the flow field that exhibit complex flow phenomena are refined, it requires less computational resources. This enables larger batch sizes during inference at high spatial resolutions while still achieving the target accuracy compared to SOTA methods for SR. To achieve this, ADARNet distinguishes between different regions of the domain by dividing the input LR flow field into fixed-size *patches*, and adaptively increases or maintains the spatial resolution of each patch based on the complexity of the flow in that region. ADARNet is an end-to-end DL-physics solver framework where the non-uniform output flow field from the model inference is input to a traditional physics solver. This coupling ensures that ADARNet meets the same convergence guarantees as AMR solvers, which is critical for practitioners [23, 24].

Specifically, ADARNet makes the following contributions, addressing the limitations of uniform SR and traditional AMR in CFD:

- **Non-uniform SR**. To enable non-uniform super-resolution, we introduce a novel *scorer-ranker-decoder* DL algorithm, where the *scorer* finds the spatial score of each patch, the *ranker* places patches in corresponding bins based on their score (which determines the target resolution of each patch), and the *decoder* reconstructs every patch in each bin to its final target resolution using semi-supervised learning.
- **Minimal user intervention**. ADARNet's training process is semi-supervised - the loss function that guides its optimization process is formed by the governing equations of the problem - which poses a two-fold advantage. First, the refinement or coarsening decisions are based on physics principles with minimal human intervention, as opposed to traditional AMR solvers that require expert, domain, and even problem-specific knowledge and a high degree of user intervention [30]. From one single training, ADARNet reaches SOTA refining/coarsening decisions for different flow problems that exhibit very different flow phenomena, showcasing remarkable generalization properties. Second, ADARNet does not require knowledge of the target resolution a priori and hence eliminates the need for expensive HR data collection.
- **Outperforms traditional AMR solvers**. We evaluate ADARNet on three canonical turbulent flows obtained from Reynolds-Averaged Navier-Stokes (RANS) simulations, considering seven flow configurations (three geometries and four boundary conditions) unseen during training. The adaptively refined meshes produced by ADARNet demonstrate excellent flow discerning properties and agreement with the baseline OpenFOAM AMR solver in refining regions of interest such as near-wall areas in wall-bounded problems (channel flow, flat plate), in flow around smooth solid bodies (flow around an airfoil), and the wake region in flow around thick solid bodies (flow around a cylinder) while maintaining less complex flow areas, such as the freestream, at low resolution. Additionally, ADARNet predicts the non-uniformly refined flow field in a single inference step, avoiding the need for multiple iterations as required by traditional AMR solvers. As a result, ADARNet achieves the same convergence guarantees as the AMR solver while accelerating it by $3.2 - 5.5\times$.
- **Reduced computational resources.** ADARNet's ability to selectively refine specific areas of the flow and avoid HR inferences across the entire domain results in a speedup of $7 - 28.5\times$ and reduces memory usage by $4.4 - 7.65\times$ at 1024×1024 spatial resolutions compared to SOTA DL methods that perform uniform SR while maintaining SOTA accuracies.

2 RELATED WORK

Adaptive Mesh Refinement (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. Different areas of the partial differential equation (PDE) domain can require different precisions where non-uniform grids are better suited. AMR algorithms dynamically identify regions that require finer resolution (such as discontinuities and steep gradients) 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. The majority of current works [7, 30] result from the early *local adaptive mesh refinement* [3], where cells are marked for refinement. Two main approaches exist for identifying cells for refinement. First, adjoint-based AMR [22], 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 [4]. Second, feature-based AMR [15], where the user supplies the 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. However, feature-based AMR approaches require both a high degree of user intervention and specific knowledge of the problem at hand and do not generalize well. Traditional iterative AMR solver are based on a handful of heuristics for re-meshing whose long-term or general optimality remains unknown. The user is expected to choose one based on their application. To overcome this limitation, Yang et al. [29] 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. In [1], the authors develop *AMRNet*, a convolutional neural network-based (CNN) model that performs *multi-resolution*, where the network outputs a uniform flow field at different resolutions. As opposed to the above approaches, we design a DL algorithm for AMR without imposing any application-specific heuristics during training. Instead, the training is guided by the fundamental governing equations of fluid dynamics – continuity and momentum equations. ADARNet's optimization process aims to reduce the residual, thereby adhering to the physics imposed by the RANS equations (see Section 3.2).

Super-resolution (SR). DL algorithms have shown impressive results for SR [14, 17]. We find SR techniques applied to both CV and CFD problems. Two main research directions exist in CV: single-image SR (SISR) and reference-based SR (RefSR). However, both SISR and RefSR have a target resolution that is both known a priori and uniform [28]. In [28], 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 [6], 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 paper, we are interested in SR, and therefore we keep all patches that cover the entire domain.

In CFD, we also find successful SR attempts. Recent works use CNNs as finite-dimensional maps [9, 19]. However, these approaches know the target resolution a priori, perform uniform SR, and require large amounts of HR labels. To eliminate the need for large amounts of HR labels, authors in [24] developed *SURFNet*, a transfer learning-based uniform SR framework. This work reduces the HR data requirement by 15× while achieving resolution invariance. However, it's also limited to uniform SR. Mesh-free, resolution-invariant methods [14, 18, 27] 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 SR. In [14], the authors developed *MeshFreeFlowNet*, an efficient framework for SR of turbulent flows that outperforms baseline models [26]. However, it lacks the ability to intrinsically discriminate between different regions of the flow, resulting in uniform output resolutions. It also suffers from the limitation of extensive HR data collection. In this paper, we present a semi-supervised DL algorithm that adaptively refines the input mesh and outputs a non-uniform HR flow field, improving both inference times and memory requirements for scaling to large problem sizes. ADARNet does not require knowledge of the target resoluton a priori, hence eliminates the need for collecting HR training data.

3 ADARNET: DL FOR AMR

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 non-uniform SR. We first detail the neural network architecture highlighting the challenges and the corresponding design choices. Then, we present the semi-supervised learning approach that leverages a hybrid loss function. Finally, we outline the end-to-end framework, which reconstructs a non-uniform HR flow field while reaching the same convergence as the state-of-the-art AMR solvers.

3.1 Neural Network Architecture

To our knowledge, no prior work has addressed non-uniform super-resolution with deep neural networks (DNNs). To tackle this problem, we design a novel DNN architecture for this problem. We decompose the problem into 3 sub-tasks: (a) identifying patches to be refined further for accuracy, (b) determining the target resolution of these salient patches, and (c) predicting the output solution in each patch given its target resolution. Figure 3 illustrates the architecture composed of a *scorer* network, a *ranker*, and a *decoder* network designed to accomplish the above 3 sub-tasks.

The DNN takes as input an LR flow field, which is first divided into fixed-size regions or *patches*. The output is a non-uniform resolution flow field, where HR is predicted only for specific patches in the domain. The RANS equations with the Spalart-Allmaras (SA) model (described in Section 4.1) 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 LR flow field consists of a four-channel tensor image where each channel

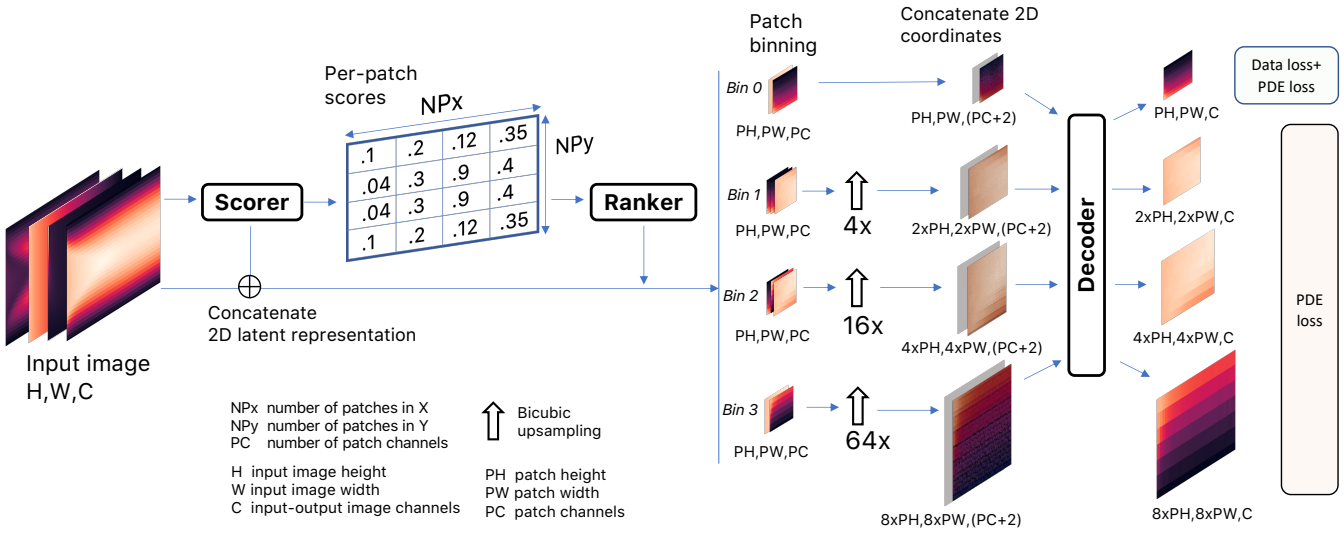

Figure 3: ADARNet's DNN. The input is a four-channel LR image where each channel represents one flow variable. The LR image is first input to the *scorer*, which divides it into patches and outputs the score of each patch. The *ranker* uses these scores to assign each patch to a bin, which is refined using a bicubic interpolation to its target resolution. Then, the refined patches are concatenated with their coordinates. Finally, the *decoder* maps this refined, 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.

represents the values of one flow variable across the entire computational grid. The DNN scores, ranks (or bins), and predicts the target resolution of each patch.

Scorer. The LR flow field is first input to the *scorer* network. This is a trainable network whose goal is to score each patch of the LR image via its 2D spatial latent representation, as illustrated in Figure 4. This network is inspired by the work of Cordonnier et al. [6] that use a similar network for finding salient patches from the input image for classification.

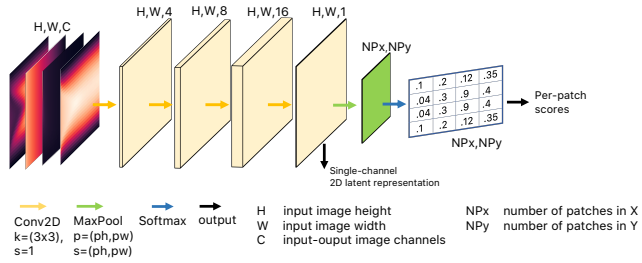

Figure 4: The *scorer* network. The first four convolutional layers extract a single-channel 2D latent spatial representation from the input LR flow field. This 2D latent representation is used to obtain the scores of each patch via a maxpooling and a softmax layer.

To identify patches for refinement, we first extract the spatial features from the input low-resolution mesh. The *scorer* network consists of a shallow CNN followed by a maxpooling layer and a softmax layer. We use convolutional layers because their inherent spatial inductive bias makes them an ideal candidate to find relevant spatial patches. The first three convolutional layers extract an abstract representation of the LR flow field. Their kernel size is $(3, 3)$ and the stride is 1. This overlap captures the spatial relationships between and among the input flow variables and maintains the spatial dimensionality. After the first three convolutional layers, we apply a single-filter convolutional layer to encapsulate the extracted spatial information in a single-channel image. This image is a 2D latent representation of the spatial dependencies in the variables of the LR 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 $NP_x \times NP_y = N$ 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. Next, 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 LR 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 it is illustrated in Figure 3. Binning consists of splitting the $0 - 1$ range of values of the scores into b bins uniformly. 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* determines the final resolution of each patch. During training, patches with the highest scores are mapped to the highest target resolution, and as the scores decrease, the target resolution is gradually reduced. Patches with the lowest scores remain in their

original LR. Since each bin (which corresponds to a specific target high-resolution) can contain a different number of patches, we need to dynamically change the batch size for each forward pass of the downstream decoder network. This differs from typical DL algorithms, where the batch size remains fixed throughout training.

After binning, we refine each patch to its target resolution using bicubic interpolation, concatenate their 2D coordinates, and then use them as input to the decoder.

Decoder. The goal of this trainable network illustrated in Figure 5 is to reconstruct the HR solution of each patch. A design choice we made in the decoder is weight sharing among resolutions as opposed to a separate decoder for each target resolution like in prior work [12]. Therefore, each patch in each bin passes through the *same* decoder, which is shared among resolutions. 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 accurately while being computationally efficient. Second, the LR patches have not been upsampled and the decoder can extract the true spatial correlations between the flow variables and coordinates in those patches. We expect this to help in recovering the true values of the HR patches. Note that the patches placed in the LR bin also passes through the shared decoder.

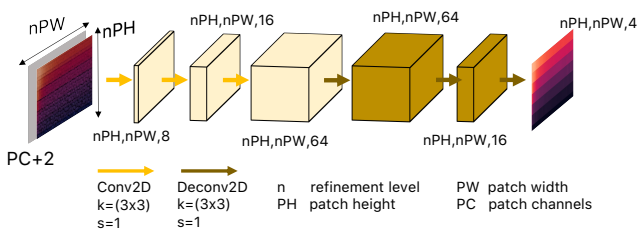

Figure 5: The *decoder* network. The input to the decoder is the intermediate patch representation concatenated with the 2D coordinates at its target resolution. This network consists of 3 convolutional layers followed by 3 deconvolutional layers.

The decoder is a 6-layer convolution-deconvolutional network. We choose this architecture because convolutions will extract a deep, abstract representation used by the deconvolution layers to reconstruct the HR output. Other works [14] use a U-net type of architecture. However, we keep the spatial dimension of the layers constant because the decoder operates on a per-patch basis and reducing the number of features that represent the patch is not desired. We use a stride of 1 to not lose any spatial information. The number of filters of the 3 convolutional layers followed by 3 deconvolutional layers are 8, 16, 64, 64, 16, 4; the kernel size is $(3, 3)$, and the stride is 1.

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.

3.2 Loss function

Since the network is not trained using the same heuristics as classical AMR solvers, we do not expect both to refine the exact same regions of the input. Stitching together the output of CFD solvers at different spatial resolutions is not feasible as this results in physically inconsistent and numerically incorrect solutions. It is also not computationally practical to run the iterative AMR solver for every intermediate mesh configuration during training to calculate the loss. As a result, we face the challenge of not having ground truth data to train the model.

To tackle the above challenge, we use a hybrid loss function for semi-supervised training. It consists of data and PDE residual loss functions, where the governing equations are imposed. Equation 1 shows our loss function.

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

The loss function $\mathbb{L}$ consists of two parts. The first term, shown on the right-hand side of Equation 1, is the data loss. It is calculated as the mean squared error (MSE) between the predicted and ground truth data for each flow variable (fv) at each cell (nc) of the LR patches (np). The ground truth data is obtained using the physics solver. The second term in the right-hand side is the L2 norm of the residual of each PDE (ne) for all cells (NC) of the output image, belonging to either low or high-resolution patches. We enforce the continuity equation and the two conservation of momentum equations, hence $ne = 3$. The gradients of the variables are computed through automatic differentiation. To constrain the PDE residual of the HR patches, we downsample them using bicubic interpolation to the lowest resolution and match the ground truth data in the downsampled space [10]. With this semi-supervised learning formulation, we avoid HR labels, an advantage over SOTA SR methods that require expensive HR training data [14, 18]. Since we do not train with ground truth data from AMR solvers, the network does not learn the solver's heuristics that have a high degree of user intervention. Therefore, ADARNet makes its own refining decisions to learn a DL-based model for AMR.

3.3 End-to-end framework

Once the network is trained and calibrated, we use it to predict the non-uniform HR 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 refining the DNN's prediction using the physics solver [23, 24].

Figure 6 illustrates ADARNet's end-to-end framework compared with the traditional AMR solver. In ADARNet, the LR flow field is input to the DNN (see Section 3.1). After inference, we feed the DNN's non-uniform output into the physics solver that drives 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

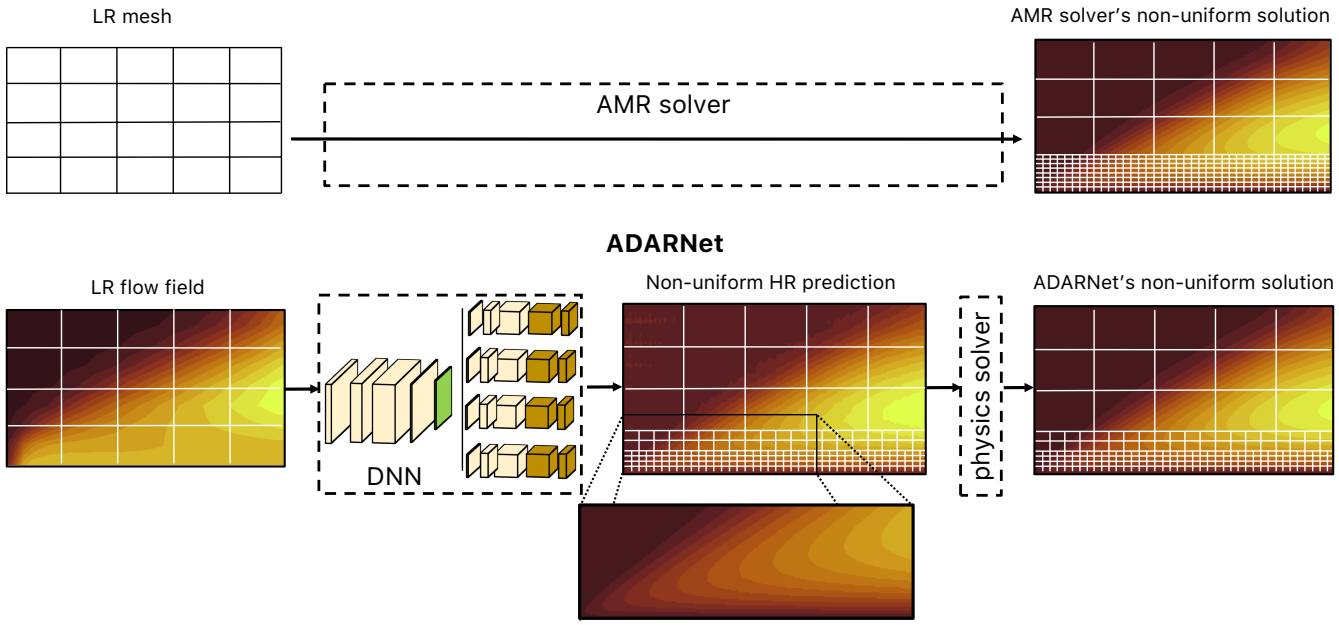

Figure 6: Top: Traditional AMR solver simulation. Bottom: ADARNet framework. After performing non-uniform SR with the DNN, we feed the output field into the physics solver, which takes the inferred solution to convergence.

final solution, convergence is accelerated. Section 5.1 empirically evaluates the performance of ADARNet against both classical AMR solvers and SOTA DL models.

4 EXPERIMENT SETUP

In this section, we present the methodology to train and evaluate ADARNet. We first present the case studies that form the dataset to train and validate ADARNet. Then, we outline the training/testing setup, parameters, and results. Finally, we describe the physics solver and the traditional AMR heuristics used for comparison.

4.1 Case studies

To train the DNN shown in Figure 3, we collect LR data from three widely studied canonical flows. The resolution of this dataset is 64×256, which is typical for LR solutions of these training cases. We use the RANS equations with the one-equation SA model to collect the data. The RANS equations describe turbulent flows as:

$$\frac{\partial \bar{U}_i}{\partial x_i} = 0 \tag{2}$$

$$\bar{U}_j \frac{\partial \bar{U}_i}{\partial x_j} = \frac{\partial}{\partial x_j}\left[-(\bar{p})\,\delta_{ij} + (\nu + \nu_t)\left(\frac{\partial \bar{U}_i}{\partial x_j} + \frac{\partial \bar{U}_j}{\partial x_i}\right)\right] \tag{3}$$

where $\bar{U}$ is the mean velocity, $\bar{p}$ is the kinematic mean pressure, ν is the fluid laminar viscosity, and ν_t is the eddy viscosity. We model ν_t with the SA one-equation model, that provides a transport equation to compute the eddy viscosity [25], described in Equation 4

$$\bar{U}_i \frac{\partial \tilde{\nu}}{\partial x_i} = C_{b1}\,(1 - f_{t2})\,\tilde{S}\tilde{\nu} - \left[C_{w1} f_w - \frac{C_{b1}}{\kappa^2} f_{t2}\right]\left(\frac{\tilde{\nu}}{d}^2\right) + \frac{1}{\sigma}\left[\frac{\partial}{\partial x_i}\left((\nu + \tilde{\nu})\frac{\partial \tilde{\nu}}{\partial x_i}\right) + C_{b2}\frac{\partial \tilde{\nu}}{\partial x_j}\frac{\partial \tilde{\nu}}{\partial x_j}\right] \tag{4}$$

From Equation 4 we can compute the eddy viscosity from $\tilde{\nu}$ as $\nu_t = \tilde{\nu} f_{v1}$. These equations represent the most popular 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). $C_{b1}, C_{w1}, C_{b2}, \kappa$, and σ are constants specific to the model, found experimentally. d is the closest distance to a solid surface. The constants of the model are those in its original reference [25].

Turbulent flow in a channel. 2D channel flow has been widely studied in the literature. A common strategy to evaluate channel flow is to vary the input velocity to the channel. This is the same as varying the Reynolds number[1]. 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 = 2e3 (when turbulent effects start to appear) to Re = 2.3e3, and then, 9700 more samples from Re = 2.7e3 to Re = 1.35e4. We leave Re = 2.5e3 and Re = 1.5e4 as our test cases. Section 5 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.

[1]The Reynolds number, or Re, is a non-dimensional coefficient that quantifies the flow conditions of the problem. $Re = \frac{UL}{\nu}$, where U is the input velocity to the problem at hand, L is a characteritic length, and ν is the laminar viscosity

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. By varying the incoming velocity we collect 10000 samples. For flat plate, incompressible turbulent effects do not appear up until Re = 1.35e5 and scale up to Re = 5e6. We collect 2000 samples from Re = 1.35e5 to Re = 2e5 and another 8000 additional samples from Re = 3e5 until Re = 1.1e6. We leave Re = 2.5e5 and Re = 1.35e6 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.

Turbulent flow around ellipses. External aerodynamics simulations are relevant for aerospace industrial applications. We gather LR 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 7 shows these configurations. 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 5e4 and 9e4. We select flow around a cylinder at Re = 1e5, flow around a symmetric airfoil (NACA0012) at Re = 2.5e4, and flow around a non-symmetric airfoil (NACA1412) at Re = 2.5e4 as test cases. 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).

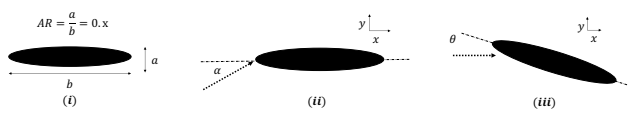

Figure 7: Ellipse geometry and characteristics (i) aspect ratio, (ii) angle of attack α, and (iii) pitch angle θ are tunable parameters.

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

4.2 Training and Testing Setup

The methodology described in Section 3 allows for multiple combinations of parameters. For instance, we can select different patch sizes or different number of target resolutions. In this section, we explain our training design choices.

First, the DNN's convolutional kernels of size 3×3 require a minimum input image size to extract relevant information. Therefore, we fix our patch sizes at $ph \times pw = 16 \times 16$, which leads to $N = 64$ total number of patches for each sample. Larger patch sizes 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 LR 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 [15] to avoid tiny computational cells. This also allows us to compare ADARNet with SOTA approaches that attempt 64× SR. The patch size and the number of bins are the same during training of the DNN and 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 [16] 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 GPUs. However, ADARNet is evaluated entirely on the CPU in this paper for a fair comparison with the AMR solver, which only supports CPU.

4.3 Physics solver and AMR solver

Once the model is trained, 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 (see Section 3.3). We use OpenFOAM's `pimpleFoam` solver as the physics solver in this paper.

As for the AMR solver to compare ADARNet with, we use the `dynamicMeshRefine` utility in OpenFOAM together with the `pimpleFoam` solver. This solver is a feature-based AMR solver [30]. 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.

Architecture and Libraries. 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 RESULTS AND DISCUSSION

After training and validating the network, we evaluate ADARNet's ability for non-uniform SR on two different use cases:

- *Same geometry but different boundary conditions.* We use ADARNet to refine the LR solution of flow on a geometry observed during training but at a different boundary condition. Here, the test flows configurations are channel flow and flat plate on interpolated (int) and extrapolated (ext) boundary conditions. For the former, we test on Re = 2.5e3 (int) and Re = 15e3 (ext). For the latter, we test on Re = 2.5e5 (int) and Re = 1.35e6 (ext).
- *Different geometry and 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 model to predict the flow around a cylinder at Re = 1e5, the flow around a symmetric NACA0012 airfoil at Re = 2.5e4, and the non-symmetric NACA1412 airfoil at Re = 2.5e4, as seen in Figure 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 4.3). Then, we present a performance analysis of ADARNet by evaluating (1) its speedup over the AMR solver and (2) the inference time and memory usage compared to a SOTA neural network model that performs uniform SR. The baseline neural network used for comparison is described in Section 5.2.

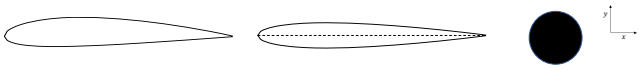

Figure 8: Non-symmetric NACA1412 airfoil (left), symmetric NACA0012 airfoil (center), and cylinder (right) as test geometries. These test cases stress the generalization capacity of ADARNet on geometries unseen-during-training.

5.1 Correctness and Accuracy of ADARNet

We first conduct a qualitative evaluation by visualizing (a) the refined/unrefined areas and (b) the final flow field by both ADARNet and AMR solver. Because of the difference in their inherent heuristics (ADARNet follows a DL-based approach, while the AMR solver follows user-given heuristics as explained in Section 4.3), 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.

Next, we conduct a quantitative comparison between the two methods using a grid convergence study [8]. 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. 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. 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.

Qualitative results. Figure 9 shows the refined/unrefined results for five test cases: channel flow at Re = 2.5e3, flat plate at Re = 1.35e6, cylinder, and the two airfoil test cases. It 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 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

[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.

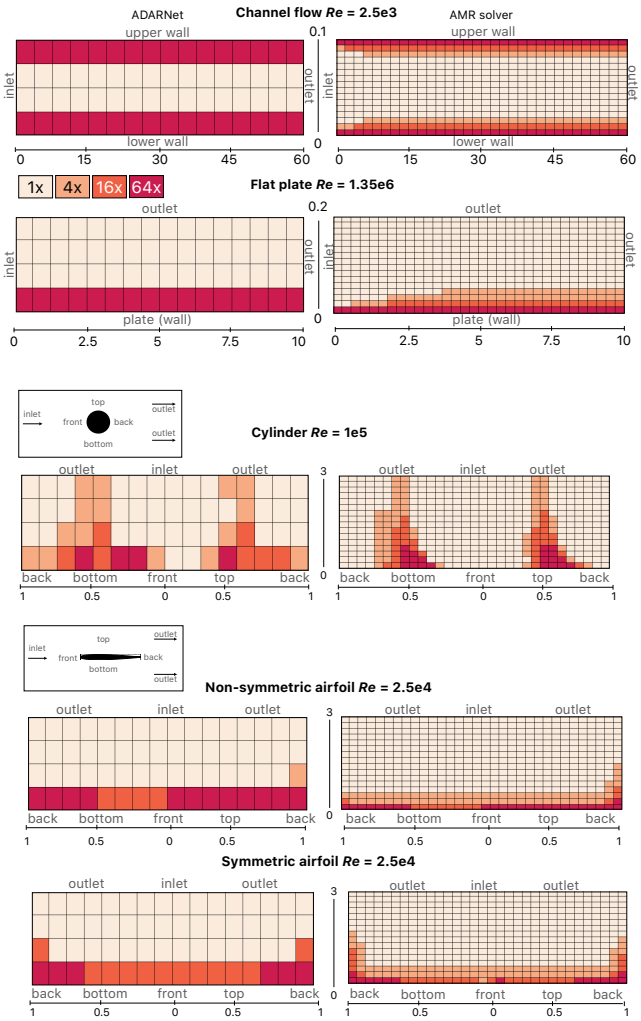

Figure 9: Per-patch fluid domain and the level of refinement of each patch for the test cases. We compare ADARNet's prediction (left) versus AMR solver's output (right). Both axes are in meters.

(first row in Figure 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 and in the symmetric airfoil case. 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 (i.e., 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. This is opposed 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 3.2) so neither dominates the other. If the data loss dominates, the network overfits to the given low-resolution data which is undesired. On the other hand, if the PDE residual loss dominates, the network yields a constant value of the flow variables in the entire domain. After a sensitive study, 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 losses reach a value of $9e-6$ 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 because 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 body.

Figure 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 3.1). 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 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 Figure 10 for the cylinder and non-symmetric airfoil test cases.

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 11 shows the value of the QoI for each test case with increasing refinement level n.

We make two main observations from the plots in Figure 11. 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 of the cylinder case in Figure 11 shows, in red, the experimental value of C_D reported in Hoerner [13], 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 [2].

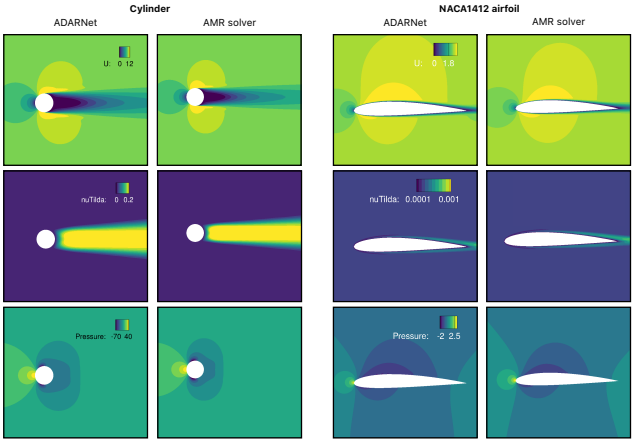

Figure 10: Comparison between ADARNet's and the AMR solver solutions for $b = 4$ levels of refinement for the cylinder and the non-symmetric airfoil cases. Velocity in $\mathrm{m\,s^{-1}}$ (top), kinematic pressure in $\mathrm{m^2/s^2}$ (bottom), and modified eddy viscosity in $\mathrm{m^2/s}$ (middle).

5.2 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 11 at $n = 3$. Recall that ADARNet's TTC is the sum of obtaining the low-resolution input image, the inference time, and the time the physics solver takes to drive the solution from inference to convergence. Table 1 shows these times and reports the iterations-to-convergence (ITC) taken by both the physics solver and the AMR solver. ADARNet achieves $3-4.5\times$ speedup for interpolated cases and $2.6-3\times$ for extrapolated cases of channel flow and flat plate.

ADARNet obtains an impressive $2.7\times$ speedup for 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 10), a region with highly nonlinear, complex flow behavior is challenging. ADARNet also speeds up both airfoil test cases. The speedup for this test cases is higher than the cylinder because the flow behavior is smoother (no flow separation). However, we observe a lower speedup for the non-symmetric (NACA1412) airfoil case compared to the symmetric one. This result is expected because the non-symmetric airfoil is an unseen-during-training geometry and has a feature (non-symmetry) that is not present on the training geometries (symmetric ellipses). 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

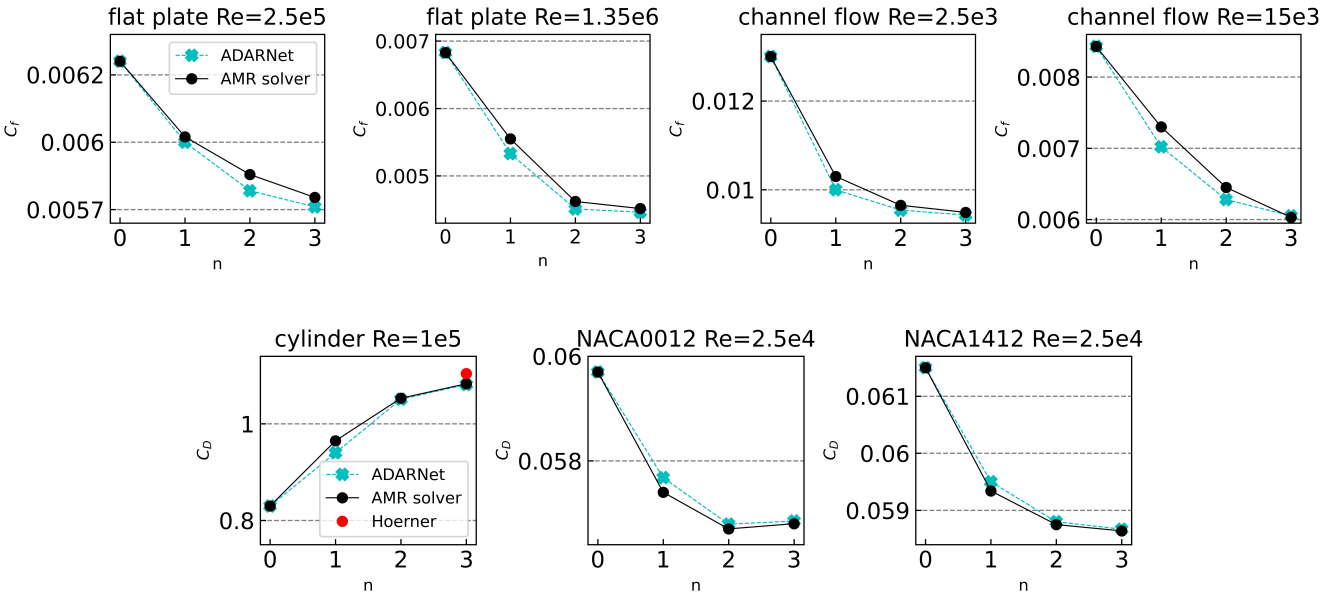

Figure 11: Value of the QoI versus refinement level n for ADARNet (blue) and the AMR solver (black) for each test case. C_f refers to coefficient of friction, and C_D to coefficient of drag. The red dot is the experimental value for the cylinder case found in [13]. Both algorithms converge as we increase the mesh refinement level from the original coarse mesh.

excellent grid convergence properties, and significantly accelerates the traditional AMR solver by $2.6 - 4.5\times$.

Table 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 obtaining the low-resolution flow field (lr), 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.

	AMR solver		ADARNet		
	ITC	TTC	ITC	TTC lr + inf + ps	Speedup
channel flow Re = 2.5e3	3369	3.0	2261	0.3 + 0.012 + 0.68	3×
channel flow Re = 15e3	4940	3.1	2022	0.3 + 0.110 + 0.80	2.6×
flat plate Re = 2.5e5	3389	2.7	1364	0.11 + 0.008 + 0.48	4.5×
flat plate Re = 1.35e6	5000	2.0	2214	0.24 + 0.009 + 0.42	3×
cylinder Re = 1e5	11155	4.8	4598	0.25 + 0.0063 + 1.50	2.7×
N0012 Re = 2.5e4	2267	2.0	1150	0.06 + 0.005 + 0.55	3.3×
N1412 Re = 2.5e4	2637	2.1	1720	0.1 + 0.005 + 0.62	2.9×

Next, we evaluate ADARNet's performance by comparing it with a baseline neural network. Recall that one of the goals of this paper is to perform non-uniform SR to avoid high-resolution inference in areas that do not require it. We hypothesize that ADARNet is advantageous over SOTA methods that perform uniform SR [9, 10, 14, 24] because these methods require 64× larger labels for 64× SR. To validate our hypothesis, we build the SURFNet [24] framework and use it as our baseline. We evaluate ADARNet and SURFNet using two metrics. First, for a 64× SR, we compare the time required to achieve the same level of accuracy. Both frameworks consist of a DNN inference followed by a physics solver that guarantees convergence requirements. Therefore, we compare the end-to-end frameworks and report the inference and convergence times. Second, we compare the memory consumption during inference. Since both models perform inference on a CPU, we report these metrics on the CPU described in 4.3. The results are presented in Table 2.

Table 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).

	Memory usage			Time (inf + ps)		
	SURFNet	ADARNet	rf	SURFNet	ADARNet	Speedup
cf Re = 2.5e3	3.9	0.88	4.4×	0.25 + 14	1.2e − 2 + 0.68	20.6×
cf Re = 15e3	3.9	0.82	4.8×	0.25 + 14.5	1.1e − 2 + 0.80	18.2×
fp Re = 2.5e5	3.9	0.62	6.3×	0.25 + 11	8e − 3 + 0.48	23.1×
fp Re = 1.35e6	3.9	0.68	5.7×	0.25 + 12	9e − 3 + 0.42	28.6×
cyl Re = 1e5	3.9	0.52	7.5×	0.25 + 10.3	6.3e − 3 + 1.50	7×
N0012 Re = 2.5e4	3.9	0.54	7.2×	0.25 + 8.35	5e − 3 + 0.55	15.5×
N1412 Re = 2.5e4	3.9	0.51	7.7×	0.25 + 8.6	5e − 3 + 0.62	14.2×

Table 2 shows that ADARNet significantly outperforms SURFNet for a 64× SR 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 $4.4 - 7.65\times$ 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 performs uniform SR.

6 CONCLUSIONS

This paper introduced ADARNet, a novel deep learning algorithm designed for non-uniform SR. The end-to-end framework predicts high-resolution accuracy only in regions of the domain where complex flow features are present, while maintaining low-resolution in areas with smooth variations. This approach significantly enhances scalability and performance, making it more practical for large-scale CFD simulations. Trained on low-resolution data from three canonical flows, ADARNet can accurately predict non-uniform flow fields for flow cases with boundary conditions or geometries unseen during training. Quantitative evaluations show that ADARNet achieves the same convergence guarantees as traditional AMR solvers, and demonstrates excellent agreement with their heuristics, while outperforming them by $2.6 - 4.5\times$. By super-resolving only regions of interest, ADARNet significantly reduces the end-to-end time and memory usage over state-of-the-art DL methods that perform uniform SR. Note that while the case study considered in this paper is solving the RANS equations coupled with a turbulence model, our approach is agnostic to the specific PDE being solved. ADARNet can be re-trained for other PDEs by changing the PDE loss, making it versatile and applicable to a wide range of simulations. Overall, ADARNet represents a significant advancement in the field of non-uniform SR and has the potential to accelerate design space exploration and optimization in CFD, paving the way for more efficient and accurate computational studies in fluid dynamics and beyond.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation under the award number 1750549. We gratefully acknowledge the computing resources provided on HPC3, a high-performance computing cluster operated by the Research Cyberinfrastructure Center at the University of California, Irvine.

REFERENCES

[1] Yuuichi Asahi, Sora Hatayama, Takashi Shimokawabe, Naoyuki Onodera, Yuta Hasegawa, and Yasuhiro Idomura. 2021. AMR-Net: Convolutional Neural Networks for Multi-resolution Steady Flow Prediction. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 686–691.

[2] Yan Bao, Dai Zhou, Cheng Huang, Qier Wu, and Xiang-qiao Chen. 2011. Numerical prediction of aerodynamic characteristics of prismatic cylinder by finite element method with Spalart–Allmaras turbulence model. *Computers & structures* 89, 3-4 (2011), 325–338.

[3] Marsha J Berger and Joseph Oliger. 1984. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of computational Physics* 53, 3 (1984), 484–512.

[4] Jan Bohn and Michael Feischl. 2021. Recurrent neural networks as optimal mesh refinement strategies. *Computers & Mathematics with Applications* 97 (2021), 61–76.

[5] Greg L Bryan, Michael L Norman, Brian W O'Shea, Tom Abel, John H Wise, Matthew J Turk, Daniel R Reynolds, David C Collins, Peng Wang, Samuel W Skillman, et al. 2014. Enzo: An adaptive mesh refinement code for astrophysics. *The Astrophysical Journal Supplement Series* 211, 2 (2014), 19.

[6] Jean-Baptiste Cordonnier, Aravindh Mahendran, Alexey Dosovitskiy, Dirk Weissenborn, Jakob Uszkoreit, and Thomas Unterthiner. 2021. Differentiable Patch Selection for Image Recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2351–2360.

[7] Anshu Dubey, Martin Berzins, Carsten Burstedde, Michael L. Norman, Didem Unat, and Mohammed Wahib. 2021. Structured Adaptive Mesh Refinement Adaptations to Retain Performance Portability With Increasing Heterogeneity. *Computing in Science Engineering* 23, 5 (2021), 62–66. https://doi.org/10.1109/MCSE.2021.3099603

[8] Luis Eça and Martin Hoekstra. 2014. A procedure for the estimation of the numerical uncertainty of CFD calculations based on grid refinement studies. *Journal of computational physics* 262 (2014), 104–130.

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

[10] Han Gao, Luning Sun, and Jian-Xun Wang. 2021. Super-resolution and denoising of fluid flow using physics-informed convolutional neural networks without high-resolution labels. *Physics of Fluids* 33, 7 (2021), 073603.

[11] UKNG Ghia, Kirti N Ghia, and CT Shin. 1982. High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *Journal of computational physics* 48, 3 (1982), 387–411.

[12] Xiaoxiao Guo, Wei Li, and Francesco Iorio. 2016. Convolutional neural networks for steady flow approximation. (2016), 481–490.

[13] Sighard F Hoerner. 1965. Fluid-dynamic drag. *Hoerner fluid dynamics* (1965).

[14] Chiyu Max Jiang, Soheil Esmaeilzadeh, Kamyar Azizzadenesheli, Karthik Kashinath, Mustafa Mustafa, Hamdi A Tchelepi, Philip Marcus, Anima Anandkumar, et al. 2020. MeshfreeFlowNet: A Physics-Constrained Deep Continuous Space-Time Super-Resolution Framework. *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)* (2020).

[15] N Kasmai, D Thompson, E Luke, M Jankun-Kelly, and R Machiraju. 2011. Feature-based adaptive mesh refinement for wingtip vortices. *International journal for numerical methods in fluids* 66, 10 (2011), 1274–1294.

[16] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[17] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. 2017. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4681–4690.

[18] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. 2020. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895* (2020).

[19] Bo Liu, Jiupeng Tang, Haibo Huang, and Xi-Yun Lu. 2020. Deep learning methods for super-resolution reconstruction of turbulent flows. *Physics of Fluids* 32, 2 (2020), 025105.

[20] Bahareh Mostafazadeh, Ferran Marti, Feng Liu, and Aparna Chandramowlishwaran. 2018. Roofline Guided Design and Analysis of a Multi-stencil CFD Solver for Multicore Performance. In *Proc. 32nd IEEE Int'l. Parallel and Distributed Processing Symp. (IPDPS)*. Vancouver, British Columbia, Canada, 753–762.

[21] Bahareh Mostafazadeh Davani, Ferran Marti, Behnam Pourghassemi, Feng Liu, and Aparna Chandramowlishwaran. 2017. Unsteady Navier-Stokes Computations on GPU Architectures. In *23rd AIAA Computational Fluid Dynamics Conference*. 4508.

[22] Marian Nemec, Michael Aftosmis, and Mathias Wintzer. 2008. Adjoint-based adaptive mesh refinement for complex geometries. In *46th AIAA Aerospace Sciences Meeting and Exhibit*. 725.

[23] Octavi Obiols-Sales, Abhinav Vishnu, Nicholas Malaya, and Aparna Chandramowliswharan. 2020. CFDNet: A deep learning-based accelerator for fluid simulations. In *Proceedings of the 34th ACM International Conference on Supercomputing*. 1–12.

[24] Octavi Obiols-Sales, Abhinav Vishnu, Nicholas Malaya, and Aparna Chandramowliswharan. 2021. SURFNet: Super-resolution of Turbulent Flows with Transfer Learning using Small Datasets. In *30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 1–11.

[25] Philippe Spalart and Steven Allmaras. 1992. A one-equation turbulence model for aerodynamic flows. In *30th aerospace sciences meeting and exhibit*. 439.

[26] Nils Thuerey, Konstantin Weißenow, Lukas Prantl, and Xiangyu Hu. 2019. Deep Learning Methods for Reynolds-Averaged Navier–Stokes Simulations of Airfoil Flows. *AIAA Journal* (2019), 1–12.

[27] Hengjie Wang, Robert Planas, Aparna Chandramowlishwaran, and Ramin Bostanabad. 2022. Mosaic flows: A transferable deep learning framework for solving PDEs on unseen domains. *Computer Methods in Applied Mechanics and Engineering* 389 (2022), 114424.

[28] Fuzhi Yang, Huan Yang, Jianlong Fu, Hongtao Lu, and Baining Guo. 2020. Learning texture transformer network for image super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5791–5800.

[29] Jiachen Yang, Tarik Dzanic, Brenden Petersen, Jun Kudo, Ketan Mittal, Vladimir Tomov, Jean-Sylvain Camier, Tuo Zhao, Hongyuan Zha, Tzanio Kolev, et al. 2021. Reinforcement Learning for Adaptive Mesh Refinement. *arXiv preprint arXiv:2103.01342* (2021).

[30] Weiqun Zhang, Andrew Myers, Kevin Gott, Ann Almgren, and John Bell. 2021. AMReX: Block-structured adaptive mesh refinement for multiphysics applications. *The International Journal of High Performance Computing Applications* (2021), 10943420211022811.