Mesh Adaptation Using Machine Learning

By

Justin Phommachanh
B.Sc., University of Kansas, 2019

Submitted to the graduate degree program in Aerospace Engineering and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

Chair: Dr. Zhi Wang

Dr. Ray Taghavi

Dr. Shawn Keshmiri

Date Defended: 27 August 2021

The thesis committee for Justin Phommachanh certifies that this is the approved version of the following thesis:

Mesh Adaptation Using Machine Learning

Chair: Dr. Zhi Wang

Date Approved: 27 August 2021

Abstract

The quality of the mesh plays an important factor in the accuracy and efficiency of computational fluid dynamics (CFD) simulations. Mesh adaptation techniques are typically computationally expensive or rely on an experienced professional to identify the locations for refinement. The focus of this research is to show that a deep neural network can learn the mesh adaptation patterns needed to accurately imitate complex mesh adaptation indicators, therefore saving computational resources by avoiding costly mathematical functions such as taking derivates and replacing them with a combination of simple matrix multiplications. For this study, two different machine learning models will be trained. The first is trained through a modification of the Larsson and Toosi error indicator by focusing only on the averaged flow field. The second model is trained using the true Larsson and Toosi error indicator based on the unsteady flow fields. In both cases, the averaged flow field serve as the input data. The necessary model parameter choices will be explained, and the final model architecture will be described. The results show that machine learning models can produce a pattern that accurately represent the original adaptation indicators using either unsteady flow fields or the averaged flow field.

Acknowledgments

I would like to acknowledge my advisor and graduate committee for all the help they have provided me along the way. I would also like to acknowledge Jeremy Ims for the work he has done in mesh adaptation. With his help, the machine learning modification to his work was possible.

Dedication

I dedicate this work to my family and to my partner who has gone through all my years of college alongside me. I would also like to dedicate this work to my dog, Bella.

Table of Contents

List of Figures

List of Tables

1 Introduction

The focus of this paper will be on the work done using machine learning but will require a brief introduction to the surrounding subjects of numerical simulations and error indicators.

1.1 Computational Fluid Dynamics

As technology continues to expand, so do the demands for a fast and efficient way to prototype how an object would behave in realistic flow conditions. CFD allows for this by numerically solving a set of equations that describe the motion of viscous fluids. These set of equations are called the Navier-Stokes's equations and are the fundamental basis for most CFD applications. CFD has been a major tool for companies as it allows for an in-depth analysis of an object in the case that an experimental test would either be impractical or too costly. The methods that exist today have shown that CFD results can match the accuracy of an experimental test to a very high degree. To solve for the qualities in a flow field, the physical domain is discretized into smaller partitions or cells which then have the mathematical models applied to. The collection of these cells is defined as the mesh and the number of cells a mesh has defines how coarse or fine it is. The accuracy of a CFD simulation greatly depends on the quality of the mesh. If the mesh is too coarse, the flow field will not be properly captured, and the accuracy of the simulation will be poor. However, if the mesh is too fine the computation cost could become so large that the simulation would be unable to produce results in a realistic time frame. Finding a balance between accuracy and computation cost usually comes from an experienced individual who will continually refine the mesh to achieve a desired accuracy. CFD has been used in a wide number of applications including aircraft, automobiles, computer cooling, and cardiovascular research. One of the most important applications of CFD simulations is its ability to resolve

turbulent flow fields. Though turbulent methods are still a hot topic of research in CFD there are some existing solutions that allow for very accurate results.

1.2 Methods to Solve Turbulent Flow

Turbulent flow has been a main topic of research in CFD. Currently, there are three main methods to solve for turbulent flow. The first and most complex is a Direct Numerical Simulation (DNS). These types of simulation usually contain such a fine mesh that they cannot be considered for practical engineering applications. Instead, DNS are often used in research and academia to provide a solution to a simple geometry that can be used as a reference. In DNS all turbulent structures are solved for directly so even low Reynold's number flows would have a hard time coming to convergence. For industrial applications where the Reynold's number is usually much higher, the simulation becomes too difficult to handle for even the most powerful computers in the world (Molland).

The second method is called Large Eddy Simulations (LES). These types of simulations distinguish between large turbulent scales and small turbulent scales by applying a low-pass filtering to the Navier-Stokes's equations. LES will then directly solve for the larger structures and only model the effect caused by the smaller structures. Since the large structures contain the majority of the turbulent energy and are responsible for most of the momentum transfer and turbulent mixing, directly solving for them leads to a high accuracy simulation. As the effects of the smaller structures are only modeled and not solved for, the computational cost of LES is reduced when comparing to DNS (Matsuno).

The third and most common type of turbulence method is Reynold Averaged Navier-Stokes (RANS). This type of simulation is used by most commercial solvers such as Star CCM+ and Ansys Fluent. These simulations completely rely on turbulence modeling to capture the effects of

turbulence (Franck). Though there are many different turbulence models that can be used for RANS, the overall accuracy of the simulation is lower when compared to LES. As shown in Figure 1, the method with the least complexity and least computational cost begins with RANS and increases in complexity and cost going to LES and DNS. To reiterate, there are no modeling effects used for DNS, only the small turbulent structures are modeled for LES, and all turbulent effects are modeled for RANS.

Figure 2 shows the solution of each type of simulation for an arbitrary nozzle case. The resolution in the turbulent wake shows dramatic degradation when moving down the pyramid of simulations. As expected, RANS shows the least accurate solution as all turbulent effects have simply been modeled. This research will focus on LES as the training data for the machine learning model was obtained through a LES solution on an unstructured, hexahedral mesh. For LES, even when using the best guidelines available, there is no guarantee that the initial mesh will lead to an accurate simulation as it is impossible for the engineer to quantify the error caused by the mesh without first obtaining a solution using the mesh (Levy).

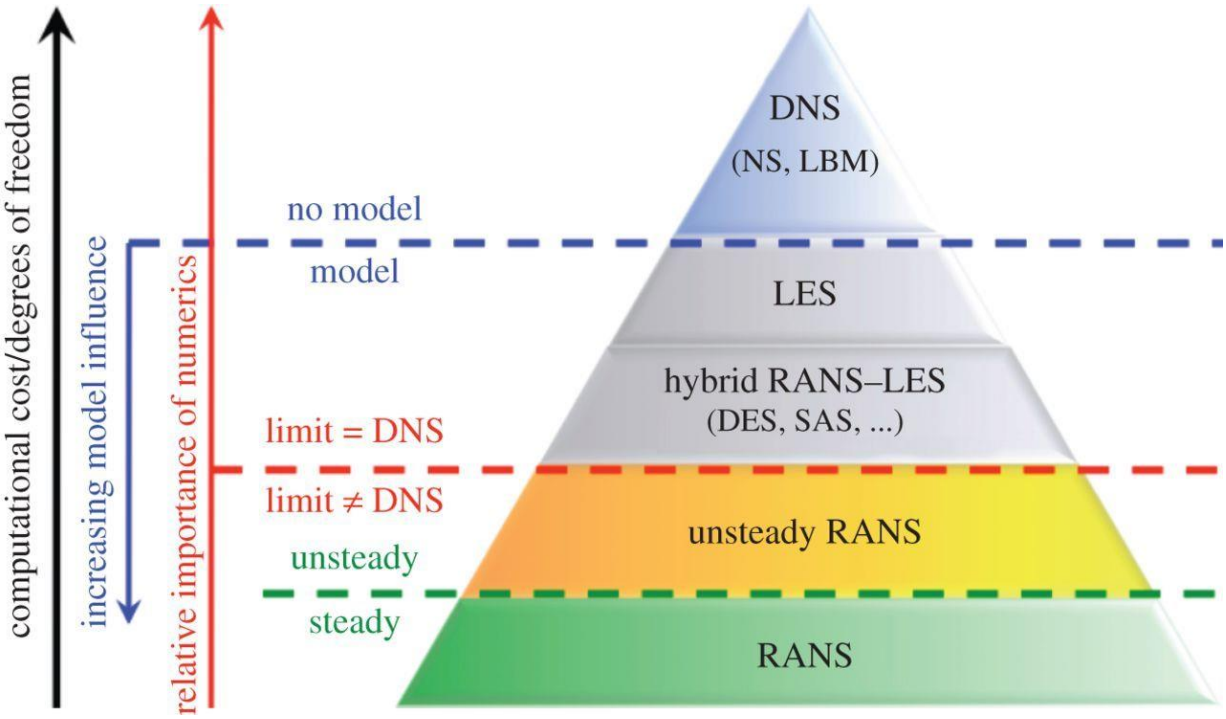

Figure 1: Different Numerical Simulations and their Complexity (Hopf)

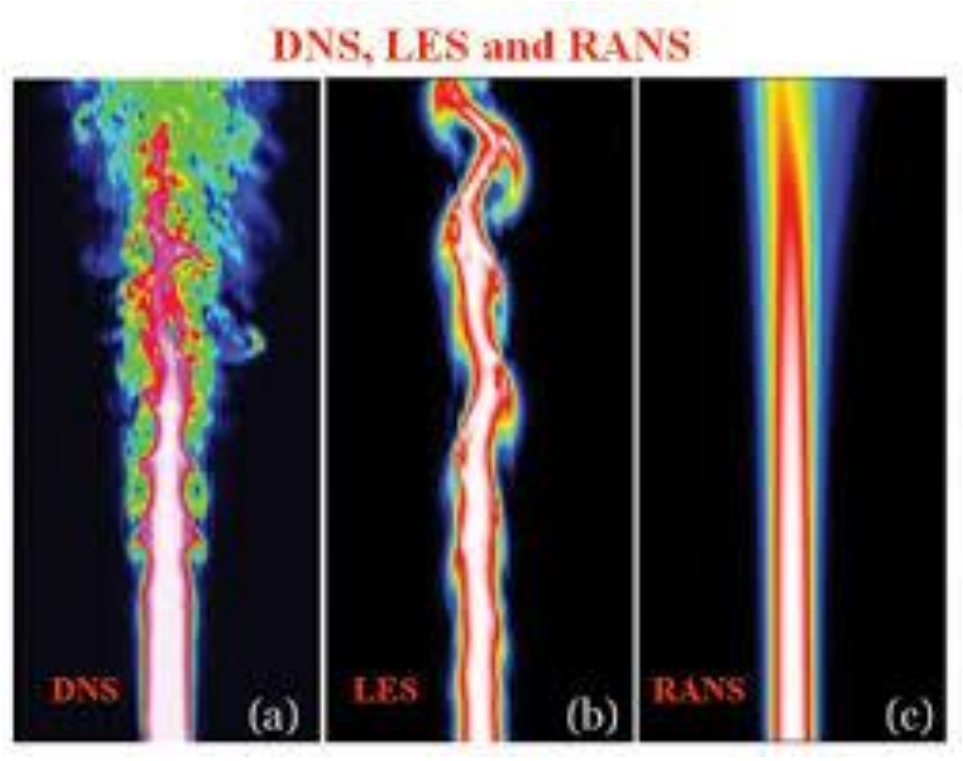

Figure 2: Example Solution Showing Difference in Turbulent Wake Between Different Simulations (Hopf)

1.3 Types of Machine Learning Models

There are three main types of machine learning models. The first is trained through supervised learning. Supervised learning provides labeled predictions along with the input data to train the model. An application of supervised learning is image recognition such training a model to identify between a dog or a cat. By training the model with labeled pictures of dogs and cats, the model will pick up on distinguishing features of both, such as the longer snout of a dog or the whiskers on a cat. Eventually, the model will learn a pattern to identify all the different defining features of both animals and gain the ability to determine whether an unseen picture is a dog or cat.

The second type of model is trained through unsupervised learning. This type of model does not have labeled predictions provided with the input data. Instead, the input data is fed into the model and the model is expected to pick up on different trends. An example of this type of model is bank fraud detection systems. Overtime, a customer's information will be fed into a system recording common transactions such as income, rent, or groceries. The model will then learn a pattern and notify the customer of any transactions outside of that pattern such as a random, large purchase.

The third and final model is trained through reinforcement learning. This type of model learns on a reward-based system as the wanted behavior is given more weighting than other options. (nvidia) describes this process similarly to training an animal. At first the animal would have no desire to listen to commands from its owner but if a treat was provided after every command the animal would become more likely to learn the command and at a faster rate compared to training without treats. A popular application of this is self-driving vehicles. The

programmer of the vehicles' system likely provided a much higher incentive for protecting the passengers and other civilians when comparing to other factors such as speed or gas mileage

The type of model used for this research is trained using supervised learning as the input data is provided with the adaptation criteria calculated from the different error indicators. The goal of this research is to show that machine learning models can learn patterns that imitate error indicators so unsupervised and reinforcement learning were not considered for this case. Further exploration could be done to see how unsupervised or reinforcement learning could be used to build a model with similar predictions.

1.4 Adaptation Methods

There are three main types of adaptation methods, h-adaptivity, p-adaptivity, and r-adaptivity. H-adaptivity is the most used method for mesh adaptation due to its great results and relative ease of use and will be the focus of this research. H-adaptivity typically utilizes an error indicator to relate the local and global error of a function. The cells with the highest influence on the global error are then picked to be refined. Error indicators will be explored more in the following section.

Unlike h-adaptivity, p-adaptivity does not change the mesh, it instead uses a higher-order polynomial to improve the accuracy in certain areas of the mesh (Li). This type of adaptation focuses more on the numerical methods of CFD and is not used for mesh adaptation.

The r-adaptation or moving mesh method redistributes the nodes or the intersection of cells, of a mesh to locations where the flow field solution has drastic changes between cells (Li; Cao). This type of adaptation does not change the total number of elements so the solution should become more accurate without an increase in computational cost. Figure 3 shows a r-adaptation

process by (Lakshmanan). Notice that only the nodes of the mesh are redistrbuted over time but the actual amount of elements do not change.

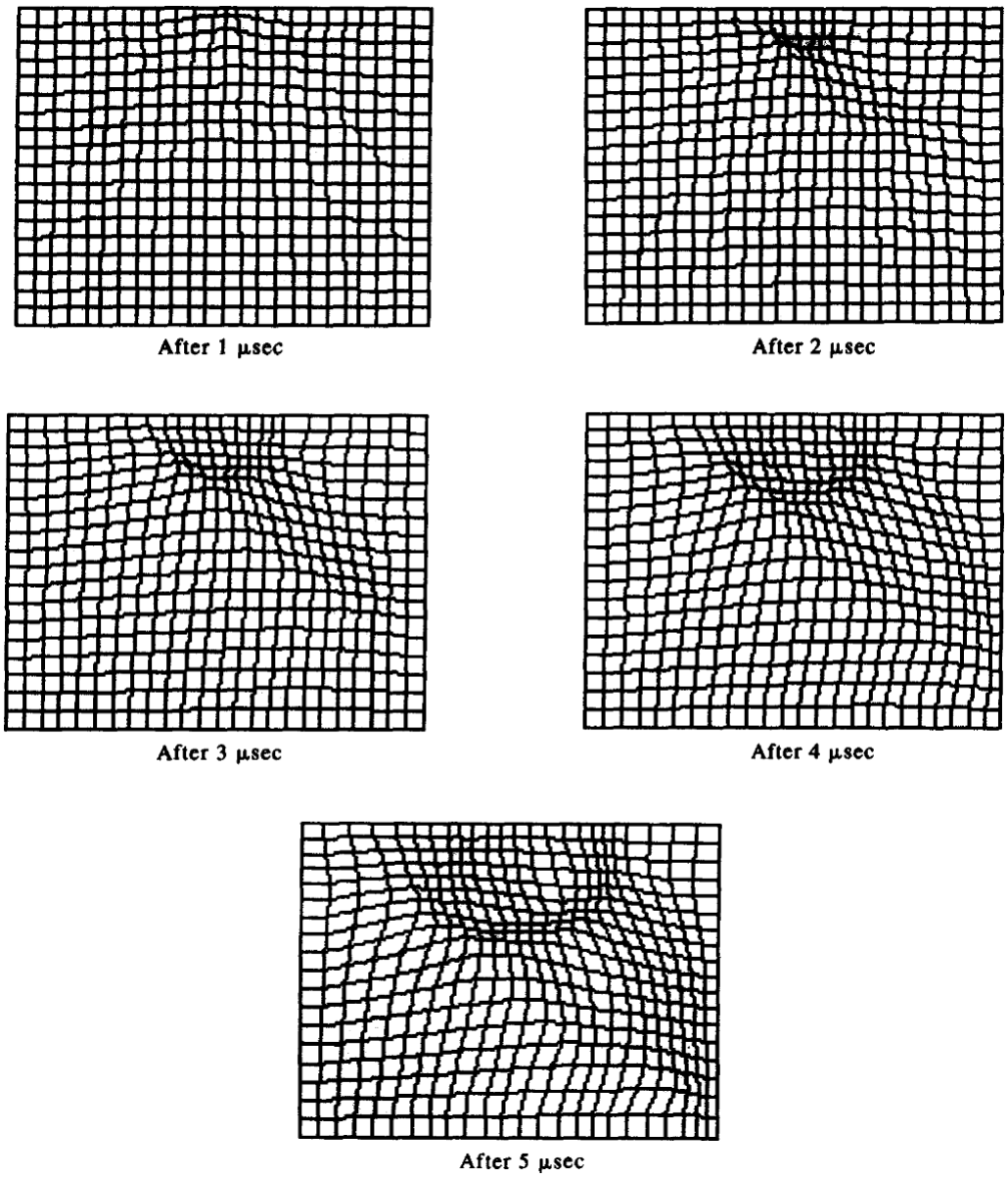

Figure 3: Example of r-adaptation

1.5 Error Indicators

Though various other adaptation criteria have been used for mesh adaptation (Frey; Gassner), for this research, error indicators are used as the criteria to determine which cells in a mesh lack refinement. There are four main types of error indicators (Roy)

1. Feature based indicators. These indicators use the gradient, curvature or smoothness of certain flow variables. They can be directional to enable anisotropic mesh adaptations.
2. Solution error or discretization error based indicators. To apply this indicator, the solution error needs to be estimated. Since it is the discretization error that we wish to reduce with mesh adaptation, on the surface it might appear the discretization error would serve as an appropriate driver for the adaption process. Reference (Roy) showed discretization error is not an appropriate mesh adaption criterion.
3. Truncation error or residual based indicators. For complex non-linear equations, the truncation error is difficult to compute. Instead, the numerical solutions or residuals on coarse and fine meshes or with different polynomial orders are used to approximate the truncation error. Furthermore, the transport of the discretization error is driven by the local truncation error (Shih).
4. Output adjoint based indicators (Fidkowski). Adjoint-based error indicators relate a specific functional output directly to the local residuals by the adjoint solution, which can be used to construct an effective error indicator to drive an adaptive procedure towards any engineering output. Such indicators can target elements which contribute the most to the output, such as the lift or drag forces.

For steady RANS flow computations, output adjoint based mesh adaptation approaches

appear to be the most efficient (Yano) in obtaining mesh-independent lift, drag, and moment coefficients. When compared to typical fixed grid solutions it is shown that the same level of accuracy can be achieved with an output adjoint based mesh adaptation at a fraction of the computational cost (Dwight; Park; Cui; Roy).

For unsteady flow problems, a similar output-adjoint approach becomes more expensive as an extra dimension in time needs to be traversed. Due to the nature of an unsteady simulation, the mesh must also be adapted at different times in the simulation to produce accurate results (Fidowski). For turbulent flow problems, the instantaneous quantities are usually very irregular and chaotic (Hu). Due to the turbulent flow, engineers typically tend to focus on the mean values of the flow field, increasing the simulation run time as time averaging can be a time-consuming process. The combination of these issues makes adjoint based mesh adaptation an unrealistic option for LES.

Instead, physics-based adaptation criteria have been developed for LES (Abbà; Antepara; Benard; Park), e.g., solution features (steep gradients or curvature), smoothness of the numerical solution either within an element or cross element boundaries, truncation errors, the smallest resolved scales.

Specifically, an adaptation criterion based on the smallest resolved scales in any coordinate direction is developed in (Toosi). The criterion allows a directional indicator in any direction to enable anisotropic mesh adaptations for mixed unstructured meshes. Furthermore, by equalizing error distributions in all coordinate directions, an “optimal” computational mesh can be obtained, which can produce a LES solution of a given resolution with minimal cost. Promising results have been demonstrated for hexahedral meshes. Below is an outline of the basic idea of an error indicator based on the smallest resolved scales which is also referred to as the Larsson & Toosi

indicator.

Let the LES solution at element *i* be U_i. An implicit low pass test filter $\hat{}$ in 1D is defined as

$$U_i = \left(I - \frac{\Delta^2}{4}\frac{\partial^2}{\partial x^2}\right)\widehat{U}_i, \tag{1}$$

where Δ is the test filter size, which can be the same as the element size. The implicit filter can be approximated by the following explicit filter

$$\widehat{U}_i = \left(I + \frac{\Delta^2}{4}\frac{\partial^2}{\partial x^2}\right)U_i. \tag{2}$$

After that, the smallest resolved scales are extracted as

$$U_i^* = U_i - \widehat{U}_i = -\frac{\Delta^2}{4}\frac{\partial^2 U_i}{\partial x^2}. \tag{3}$$

This idea can be generalized to compute the smallest resolved scales in any direction of unit vector **n**

$$U_i^{*,(\boldsymbol{n})} = U_i - \widehat{U}_i^{(n)} = -\frac{\Delta_n^2}{4}\frac{\partial^2 U_i}{\partial n^2} = -\frac{\Delta_n^2}{4}\boldsymbol{n}^T(\nabla\nabla^T U_i)\boldsymbol{n}. \tag{4}$$

Finally, the anisotropic error indicator in direction **n** is defined as

$$\mathcal{A}(\boldsymbol{n}) = \sqrt{\langle\left(U_i^{*,(n)}, U_i^{*,(n)}\right)\rangle}, \tag{5}$$

where $(\cdot,\cdot)$ denotes an inner product, while $\langle\cdot\rangle$ indicates time or phase averaging (Wang). Two error indicators are used in this research to show that machine learning can be used to learn a pattern that imitates complex error indicators. The first error indicator can be called the Larsson average indicator. It is a modification to the Larsson and Toosi indicator that focuses on the average flow field compared to the unsteady flow field. The second error indicator is the true Larsson and Toosi error indicator that uses the unsteady flow field and is shown as an example above. These error indicators are chosen as they performed the best out of a batch of several other error indicators including a residual based indicator and a smoothness based indicator.

1.6 Mesh Adaptation Process

Below is the adaptation process used to create the input data for the machine learning model. For this case, the fraction of elements to refine were the top 20% of cells that had the highest contribution to the global error.

- Run the simulation until the flow reaches a statistical “steady state”
- Compute of the adaptation criteria at a user-specified frequency and duration
- Either compute the average or maximum criteria
- Select a fraction of the elements to refine
- Produce a new mesh and restart file
- Repeat until convergence

1.7 Importance of Machine Learning in Mesh Adaptation

As stated above, computational cost is one of the biggest factors in CFD. With the additional accuracy that a LES provides it also introduces a factor of complexity that increases its computational cost when compared to RANS. With the need to resolve turbulent flow to a higher accuracy, many methods have been developed to improve LES. One of the most promising methods being mesh adaptation using solution based error indicators. Though this method shows an increase in accuracy of LES, the calculations to determine which cells need refinement are costly and must be done repeatedly until the simulation reaches convergence. These calculations can drive up the computational cost on an already costly method to resolve turbulent flow. With the use of machine learning a model can be trained that imitates the purpose of the error indicators. This would drastically reduce the computational cost of the simulation by converting the expensive functions of the error indicators down to matrix multiplications. In this way, the

improvement in accuracy from the mesh adaptation using error indicators does not have to be met with an increase in computational time.

2 Mesh Adaptation and Numerical Method

2.1 Related Work in Mesh Adaptation

Mesh adaptation techniques have been successfully applied to RANS for years, shown in (Jones; Nemec; Peter). However, mesh adaptation techniques, specifically h-adaptation for LES have been rare to come across. Due to the rising popularity of LES, relatively new work has been published that try to implement similar ideas of mesh adaptation from RANS to LES. For structured mesh adaptation for LES, (Antepara) proposes a method that uses criteria based on the variational multi-scale (VMS) decomposition theory. They can execute their adaptation method in parallel or on multiple CPU/GPU's at once and achieved major speed ups in simulation run time a long with an increase in accuracy. For unstructured meshes for LES, (Bernard; Gou) used a two criteria method for mesh adaptation. The first was to ensure a correct discretization of the mean field and the second was to ensure explicit resolution of the turbulent scale's motions. A newer approach for mesh adaptation for LES with unstructured meshes has been proposed by (Daviller) that bases their refinement criteria on the averaged kinetic energy dissipation rate. This method is then expanded by (Odier) by adapting the mesh for wall-modeled, unstructured, complex turbomachinery flows. Both showed results with high accuracy with the work done by (Odier) being quite impressive as the geometry was so complex.

As the accessibility of machine learning is relatively new, the work surrounding mesh adaptation using machine learning is sparse. At the time of writing this, almost all work has come from the past year. There have been some variations of the work proposed such as (Wu) where the authors train a model based on the distance from the nodes of the mesh to the interesting flow field. The model then adapts the mesh by moving the nodes closer to those features. They showed very promising results, but the methodology is quite different as the work

in this paper is focused on mesh adaptation using error indicators. A closer variation of the work presented in this paper is (Fidkowski). Here they train a model to determine the optimal anisotropy in a mesh. The network predicts the element aspect ratio from features of the primal and adjoint solutions. However, the sizing of the element is still based on an error estimator. Though they achieve similar goals such as a reduction in computation and implementation cost, the focus of the paper was not learning the patterns of error estimators. Another interesting variation of this research is done by (Yang). Here they choose to avoid any training based on error estimators and instead establish a model trained using reinforcement learning on known functions to generalize mesh adaptation. They state that their model can be competitive with error estimators and generalize to larger, more complex, and unseen problems.

To the best of our knowledge there has not been any research done in mesh adaptation using machine learning using a LES simulation with an unstructured, hexahedral mesh in three dimensions for the training data.

2.2 Spatial Discretization

The LES solver is based on the Flux Reconstruction (FR) method, also known as the Correction Procedure via Reconstruction (CPR) method. This method was originally developed by (Huynh) in 2007 for hyperbolic partial differential equations. One of the benefits of the method is that it can solve on mixed, unstructured meshes (Wang; Haga). This method belongs to discontinuous finite element methods, which are similar to both the discontinuous Galerkin (DG) (Cockburn) and spectral difference (Liu) methods. Here we present an introduction of the FR/CPR method by starting from a hyperbolic conservation law governing inviscid flow with initial and boundary conditions.

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{U}) = 0, \tag{6}$$

The vector **U** consists of the conservative variables, and **F** is the flux. By discretizing the computational domain with non-overlapping elements, and introducing an arbitrary test function φ in each element, the weighted residual formulation of Eq. (6) on element V_i can be expressed as

$$\int_{V_i} \left[\frac{\partial \mathbf{U}}{\partial \mathrm{t}} + \nabla \cdot \mathbf{F}(\mathbf{U})\right] \varphi \mathrm{d}\Omega = 0. \tag{7}$$

The conservative variables inside one element are assumed to be polynomials and are expressed by nodal values at certain points called solution points (SPs). After applying integration by parts to the divergence of flux, replacing the normal flux term with a common Riemann flux F_{com}^{n} and integrating back by parts, we obtain

$$\int_{V_i} \frac{\partial \mathbf{U_i}}{\partial \mathrm{t}} \varphi \mathrm{d}\Omega + \int_{V_i} \varphi \nabla \cdot \mathbf{F}(\mathbf{U}_i)\, \mathrm{d}\Omega + \int_{\partial V_i} \varphi [F_{com}^{n} - F^{n}(\mathbf{U}_i)]\, \mathrm{dS} = 0. \tag{8}$$

Now the common Riemann flux can be computed with a Riemann solver

$$F_{com}^{n} = F_{com}^{n}(\mathbf{U}_i, \mathbf{U}_{i+}, \mathbf{n}), \tag{9}$$

where $\mathbf{U_{i+}}$ stands for the solution outside the current element, and **n** denotes the outward normal direction of the interface. The normal flux at the interface is:

$$F^{n}(\mathbf{U}_i) = \mathbf{F}(\mathbf{U}_i) \cdot \mathbf{n}. \tag{10}$$

Note that if the face integral in Eq. (8) can be transformed into an element integral then the test function will be eliminated. To do so, a "correction field", δ_i is defined in each element as

$$\int_{V_i} \varphi \delta_i \, \mathrm{d}\Omega = \int_{\partial V_i} \varphi [F^{n}]\, \mathrm{dS}, \tag{11}$$

where $[F^{n}] = [F_{com}^{n} - F^{n}(\mathbf{U}_i)]$ is the normal flux jump. Combining Eqs. (8) and (11) result in

$$\int_{V_i} \left[\frac{\partial \mathbf{U_i}}{\partial \mathrm{t}} + \nabla \cdot \mathbf{F}(\mathbf{U}_i) + \delta_i\right] \varphi \mathrm{d}\Omega = 0. \tag{12}$$

The final formulation for each solution point *j* is

$$\frac{\partial \mathbf{U}_{i,j}}{\partial \mathrm{t}} + \Pi_j[\nabla \cdot \mathbf{F}(\mathbf{U}_i)] + \delta_{i,j} = 0, \tag{13}$$

where Π_j denotes a projection to the polynomial space, and the subscript j denotes the j-th solution point in a certain element.

For the viscous fluxes involving the gradient of conservative variables, we use the Bassi-Rebay 2 scheme (Bassi).

2.3 Time Integration

In the following research, only the three-stage SSP Runge-Kutta scheme (RK3) (Shu) is considered to reduce the number of factors affecting the relative performance. Let the semi-discretized scheme be written as

$$\frac{\mathrm{d}\mathbf{W}}{\mathrm{dt}} = \mathfrak{R}(\mathbf{W}), \tag{14}$$

where $\mathbf{W}$ denotes the global discrete degrees-of-freedom, and $\mathfrak{R}$ is the residual operator including the space discretization. The RK3 scheme can be written as

$$\mathbf{W}^{(1)} = \boldsymbol{W}^n + \Delta t \mathfrak{R}(\boldsymbol{W}^n) \tag{15}$$

$$\mathbf{W}^{(2)} = \frac{3}{4}\boldsymbol{W}^n + \frac{1}{4}\left[\boldsymbol{W}^{(1)} + \Delta t \mathfrak{R}(\boldsymbol{W}^{(1)})\right] \tag{16}$$

$$\mathbf{W}^{n+1} = \frac{1}{3}\boldsymbol{W}^n + \frac{2}{3}\left[\boldsymbol{W}^{(2)} + \Delta t \mathfrak{R}(\boldsymbol{W}^{(2)})\right] \tag{17}$$

where Dt is the time step (Wang, Relative Performance of High-Order over 2nd Order Schemes for Benchmark LES Problems).

3 Machine Learning Adaptation Methods

This section covers how the input data is received, the type of machine learning model used, the effects that hyperparameters have on the model and lists the model and hyperparameters used for this research.

3.1 Input Data

The input data is a combination of the cross-cell face distances and the average velocity field extracted from each solution point. The cross-cell distances are simply the distance from the middle of a face to the middle of the opposing face. An example of solution points and cross-cell distances are shown on in Figure 4, though the exact location of the points varies as the actual points are at gaussian quadrature points clustered near the boundary. Though only nine points are shown to illustrate the idea in two-dimensions the actual amount of solution points per cell is 27 due to the polynomial order of the simulation being second. From each solution point the average velocity in x, y, and z are extracted leading to a total of 84 inputs per cell.

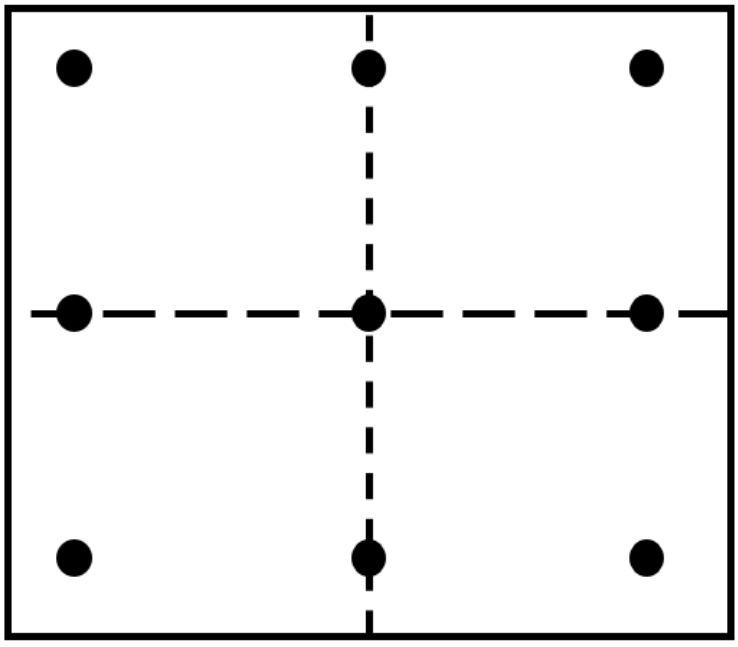

Figure 4: Illustration of Input Data in Two Dimensions

The given input data is further manipulated by scaling all inputs down to a range from -1 to 1 to help the model better predict trends. The input data is then split with 70% of the data being dedicated to training, 20% dedicated to testing, and the remaining 10% of the data being dedicated to validation. It is important to note that only the training data is responsible for the update in weights. The validation data is used to ensure the model is predicting the same way on a separate data set during training. While the testing data is reserved as a data set that the model has never seen to be tested after training.

3.2 Model Type

The model chosen is a fully connected Feed Forward Neural Network (FFNN) but with the addition of multiple hidden layers and multiple neurons, is more commonly called a Multi-Layered Perceptron (MLP) model. MLP models are a subset of supervised learning where the input data is fully processed through the network in only one direction. At the beginning of training, each layer has a set of randomized weights and biases. Each neuron receives the input from every neuron in the previous layer which it then multiplies by the weights of the layer. The neuron then takes the summation of the multiplication with the addition of the biases and feeds that into an activation function to make it a useable value for the next layer. This process continues throughout all the hidden layers until it reaches the end of the output layer. At the end of the output layer the loss is calculated, and the model is backpropagated to adjust the weights per layer until the desired loss value is achieved through the stochastic gradient descent method (El-Banbi). Figure 5 shows a basic outline of a MLP where the connections between nodes, input layers, hidden layers, and output layers can easily be seen.

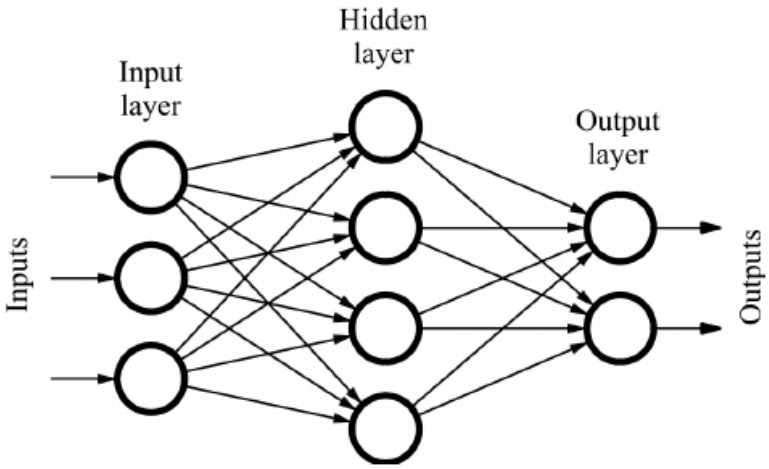

Figure 5: Example Model Showing Simple Model Architecture (Ramon)

3.3 Effect of Hyperparameters

Hyperparameters are the adjustable options of a machine learning model that control how fast and accurately the model learns. In this section multiple hyperparameters will be defined and their effect on the model will be discussed.

3.3.1 Activation Function

Activation functions are used at the end of each hidden layer to convert the output into a useable input for the next layer. An activation function also introduces non-linearity to the model, allowing it to learn complex patterns (Rios). The activation function for this research is chosen as the LeakyReLU function defined as $f(x) = \max(\alpha x, x)$ with a value of $\alpha = 0.03$. This activation function varies from the more popular ReLU function as it allows for negative inputs by converting them into small positive gradients. The ReLU function does not allow for negative inputs which limits the learning of the neurons.

3.3.2 Number of Hidden Layers

The ability for the model to learn complex patterns directly depends on the number of hidden layers it has. When a model has no hidden layers, it is only capable of representing linear separable functions or decisions. With one hidden layer the model can approximate any function

that contains a continuous mapping from one finite space to another. With two hidden layers the model can represent any arbitrary decision to any accuracy with rational activation functions (Brownlee). Though most models can be trained accurately using only two hidden layers, additional layers can improve both accuracy and reduce training time. In the past, most recommendations listed two hidden layers as the optimal amount but with the emergence of deep neural network training tools it has become relatively easy to train models with more than two hidden layers (Witten).

3.3.3 Number of Neurons

The number of neurons directly affects the model's ability to learn. The two problems associated with the number of neurons is underfitting or overfitting. Underfitting occurs when the model architecture is too small to accommodate for the complex patterns of the data set. As a result, the model will fail to predict any meaningful pattern. Overfitting occurs when the model architecture is too large for the relatively smaller data set. This causes the model to create patterns that do not actually exist in the data. Unfortunately, there is no hard rule to establish the number of neurons or hidden layers that a model should have. Instead, these parameters are usually tweaked until the desired accuracy is achieved. Figure 6 shows how the mathematical process is handled inside of one neuron.

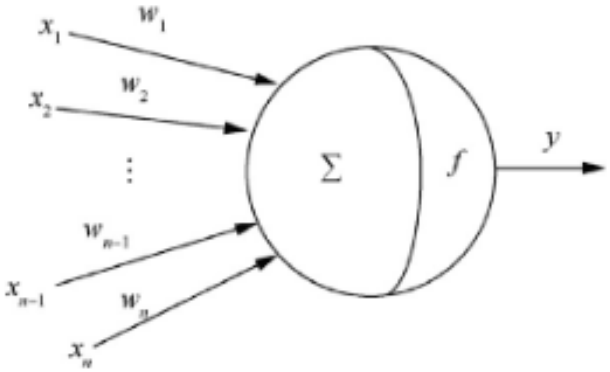

Figure 6: Neuron Diagram Showing the Connections and Functions (Wu)

3.3.4 Batch Size

Batch size is a hyperparameter that changes the number of samples that the network reads at one time. An epoch is defined when the input data is propagated through the network and the weights are updated. By defining a batch size, the input data is split into smaller samples. This allows the network to update the weights multiple times per epoch leading to a faster convergence of the model. Using a batch size also allows the model to better generalize trends as it must infer between the smaller sample sets. Figure 7 shows how an ordinary data set can be split into smaller batches which can then be fed one at a time into the model.

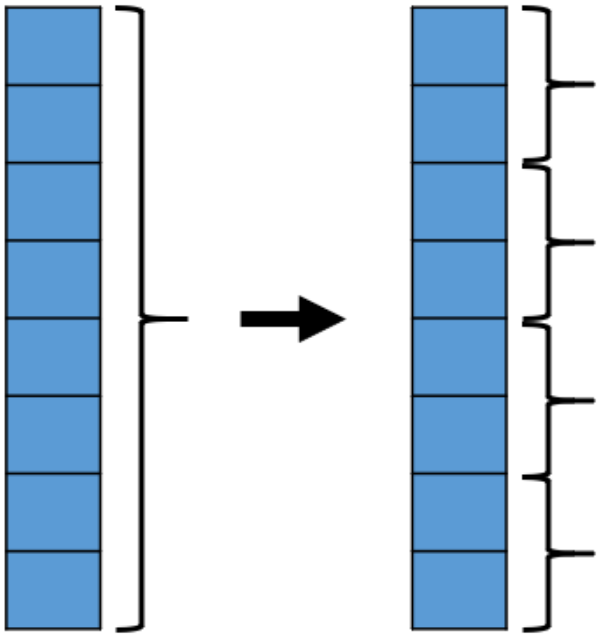

Figure 7: Illustration of Separating Input Data into Smaller Batches

3.3.5 Learning Rate

The learning rate is often considered the most important hyperparameter. It is defined as the frequency at which the weights are updated during training. Too large of a learning rate will cause the model to converge toward a false solution and too small of a learning rate will lead to an inability for the model to learn anything within a realistic timeframe. Figure 8 shows a case

study illustrating how different learning rates will affect the model. On the y-axis is accuracy and on the x-axis is the number of epochs trained for. The blue curve represents the training accuracy, and the orange curve represents the validation accuracy. It is shown that too low of a learning rate did not allow the model to learn anything in the given number of epochs. The learning rate of 1.0 caused the model to converge to a poor pattern causing volatile accuracy. The best learning rate seems to be 0.1 as the training and validation accuracy closely align and slowly come to a solution that has relatively high accuracy.

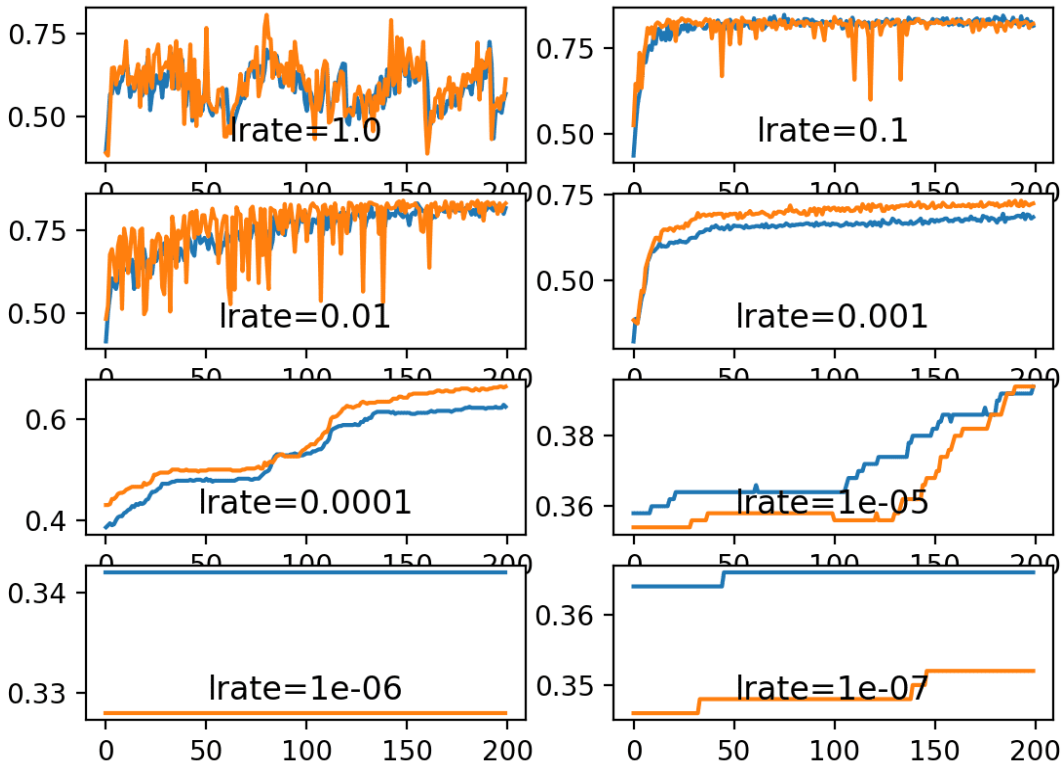

Figure 8: Difference in Learning Rate Case Study (Brownlee)

3.4 Model Structure and Parameters

The model used for this research was obtained by testing several different models. The factors talked about above such as underfitting or overfitting were avoided by adjusting the number of neurons and hidden layers until the desired accuracy was achieved. Below is a table showing the number of hidden layers and number of neurons per layer for this test case. A batch size of 10 samples were used with learning rate of 1e-5. Each model was trained for 3,000 epochs.

Table 1: Model Architecture

Layer	**Number of Neurons**
First	256
Second	128
Third	64
Fourth	32

4 Results

The results shown are for a T106-C airfoil with the flight conditions shown in Table 2. The results were gathered by exporting the weights and biases of the model and running the matrix multiplication on the base mesh. These results are then compared with the mesh achieved by the mesh adaptation formulas. A DNS solution of the airfoil is also available to make comparisons to. The results are separated into the Larsson average model and the Larsson and Toosi model. For the cell adaption figures, the red cells indicate the need to be refined, the white cells are refined due to its neighbor cells being refined, and the blue cells do not need refinement.

Table 2: Flight conditions for the T106-C

Flight condition	Value
Exit isentropic Mach	0.65
Reynolds number (based on chord)	80,000
Angle of Attack	32.7°

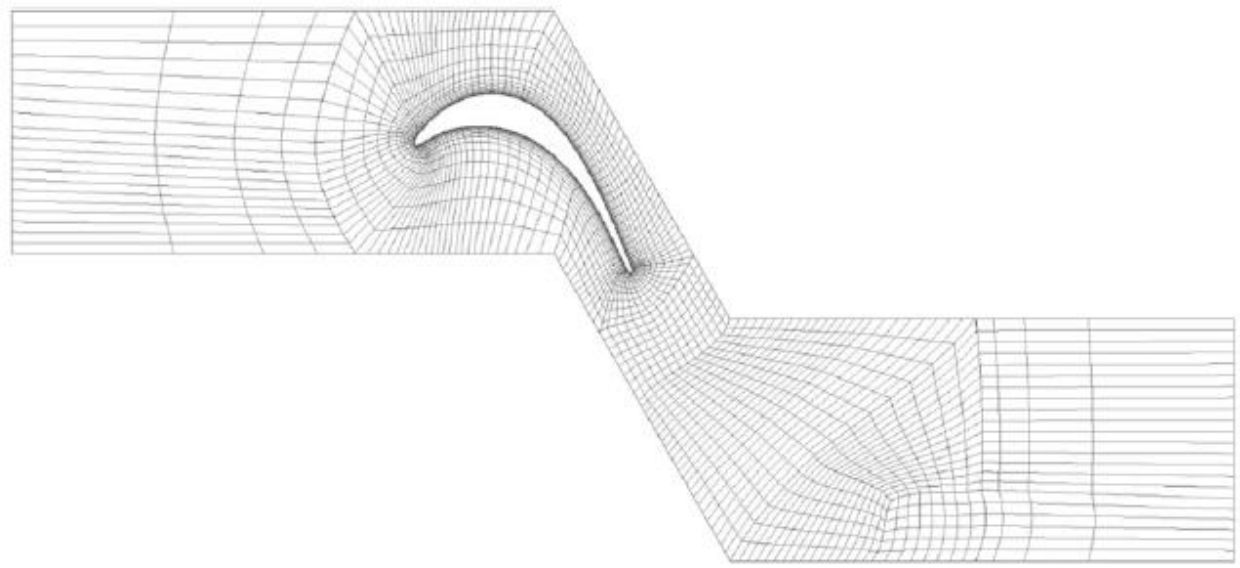

Figure 9: Base Mesh for the T106-C Airfoil

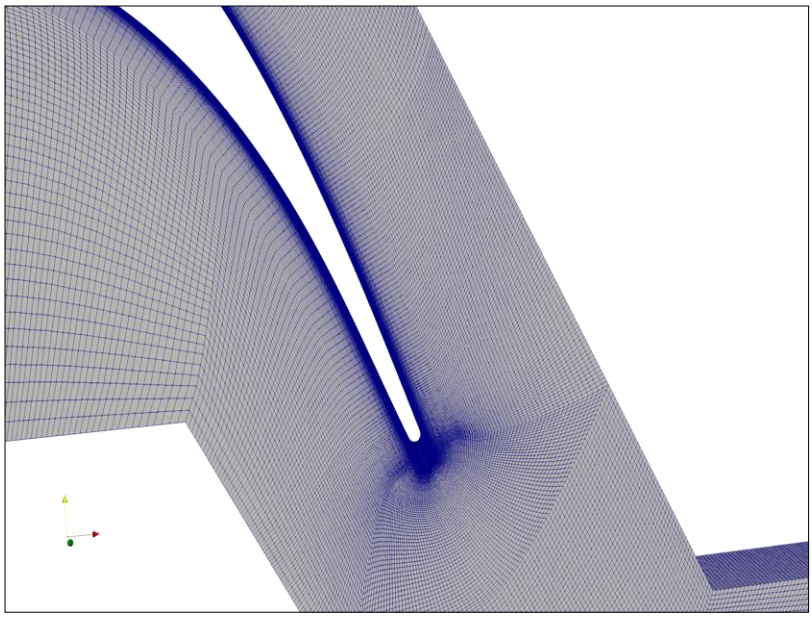

Figure 10: DNS Mesh for the T106-C Airfoil

4.1 Larsson Averaged Based Indicator

From Figure 11, the Larsson averaged model had a final training loss value of 1.79e-5 and a validation loss value of 4.25e-5. Reducing the loss by five orders of magnitude shows both convergence of the model as well as high accuracy. It is important to note that the training and validation losses overlap each other, verifying that the model was properly trained. Figure 12 is another form of verification that the model has achieved high accuracy. The test data predictions are plotted against the test data truths as a graphic way to verify the model. Ideally, a perfect diagonal line would indicate 100% accuracy.

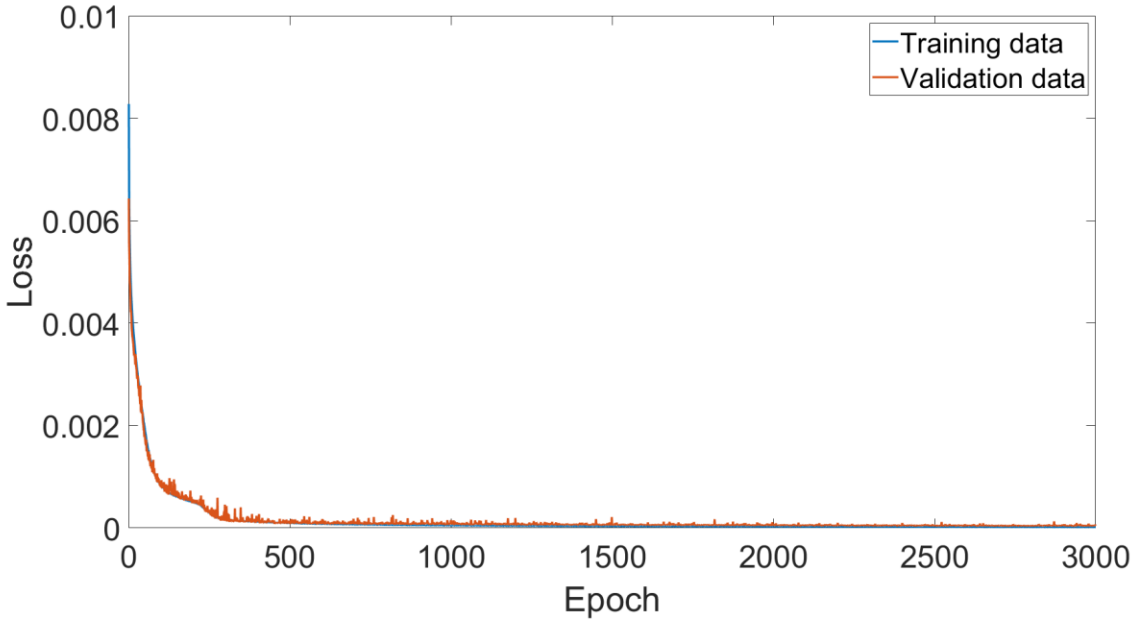

Figure 11: Loss vs Epoch for Larsson Average Indicator Model

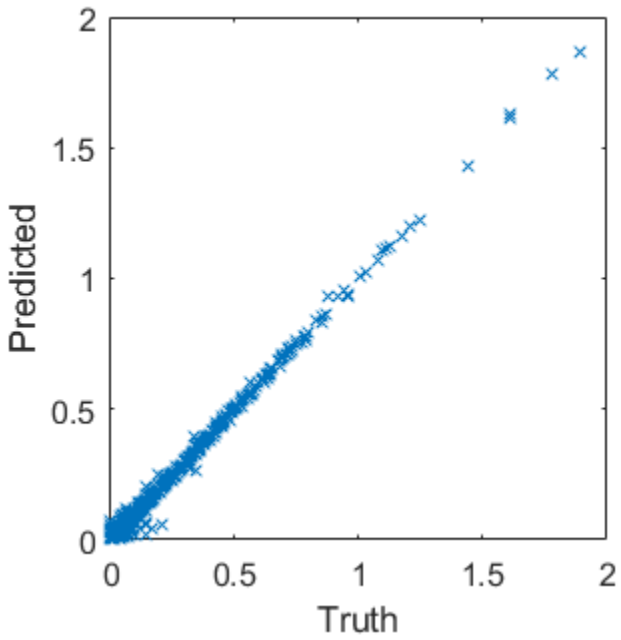

Figure 12: Predicted vs Truth for Larsson Average Indicator Test Data

The comparison between the cells chosen to adapt between the formulas vs the machine learning model are almost unnoticeable. All the important cells near the airfoil surface and turbulent wake are chosen to adapt in both cases.

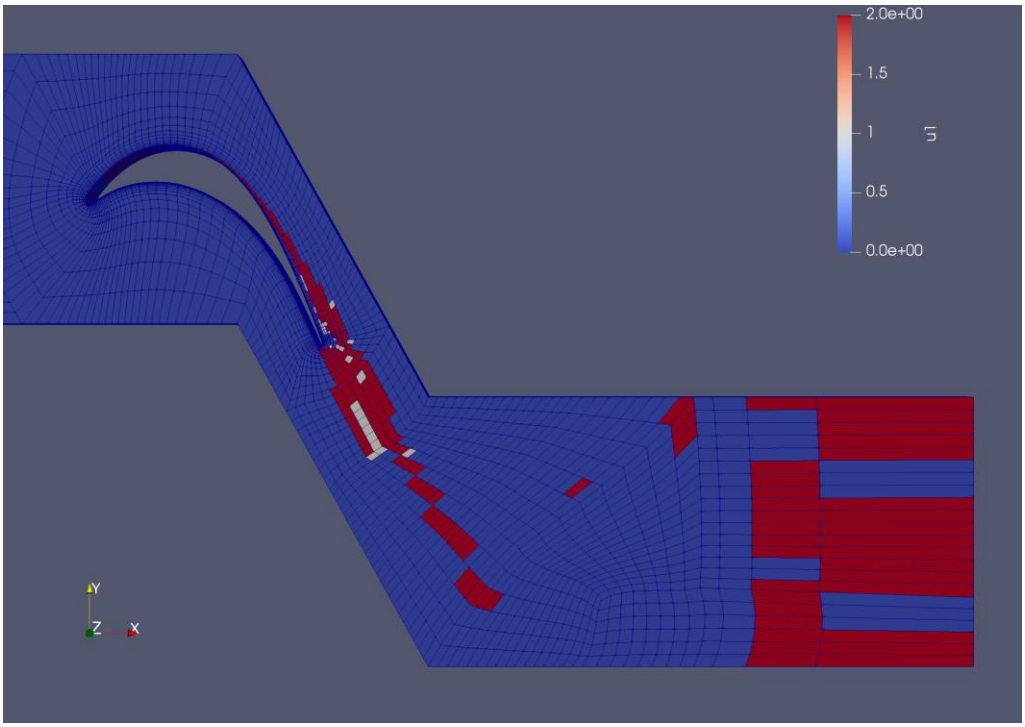

Figure 13: Cells Refined by Larsson Average Indicator Formulas

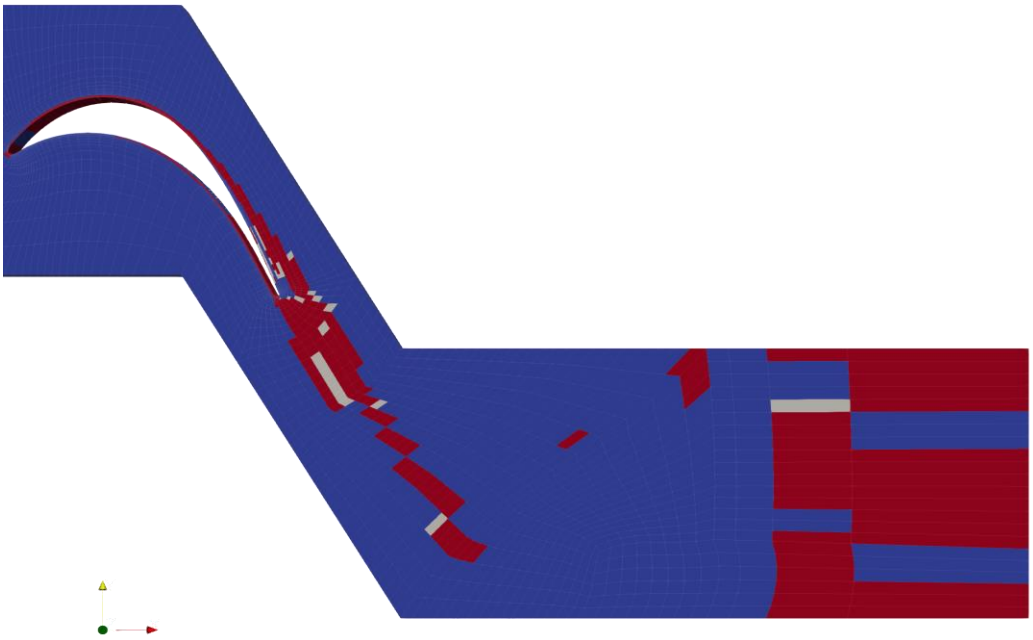

Figure 14: Cells Refined by Larsson Average Indicator Machine Learning Model

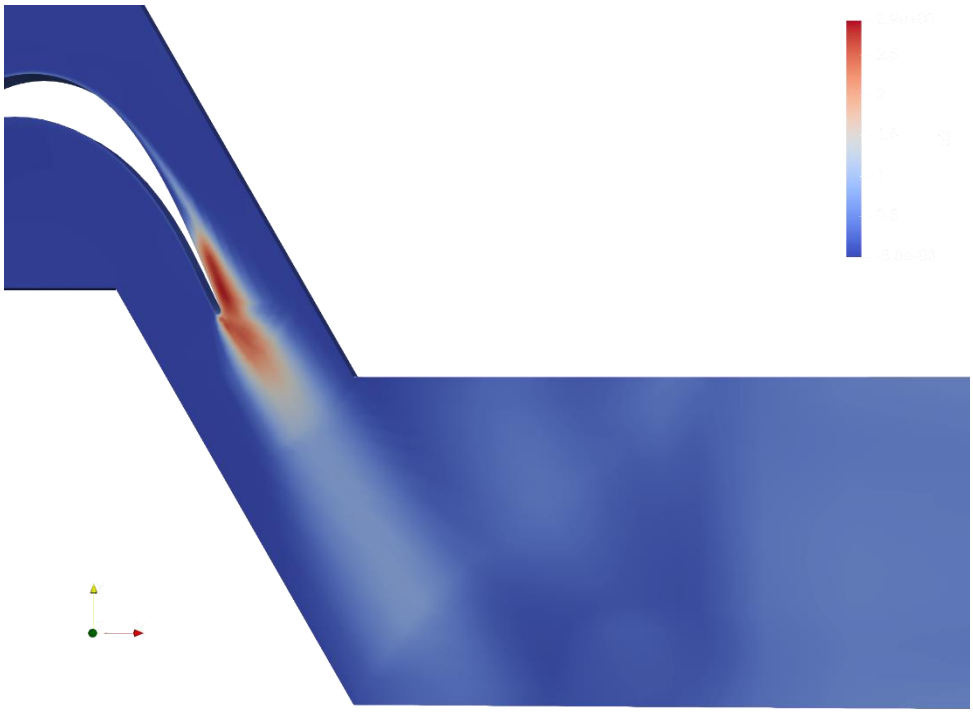

Figure 15: Raw Error Values by Larsson Average Indicator Formulas

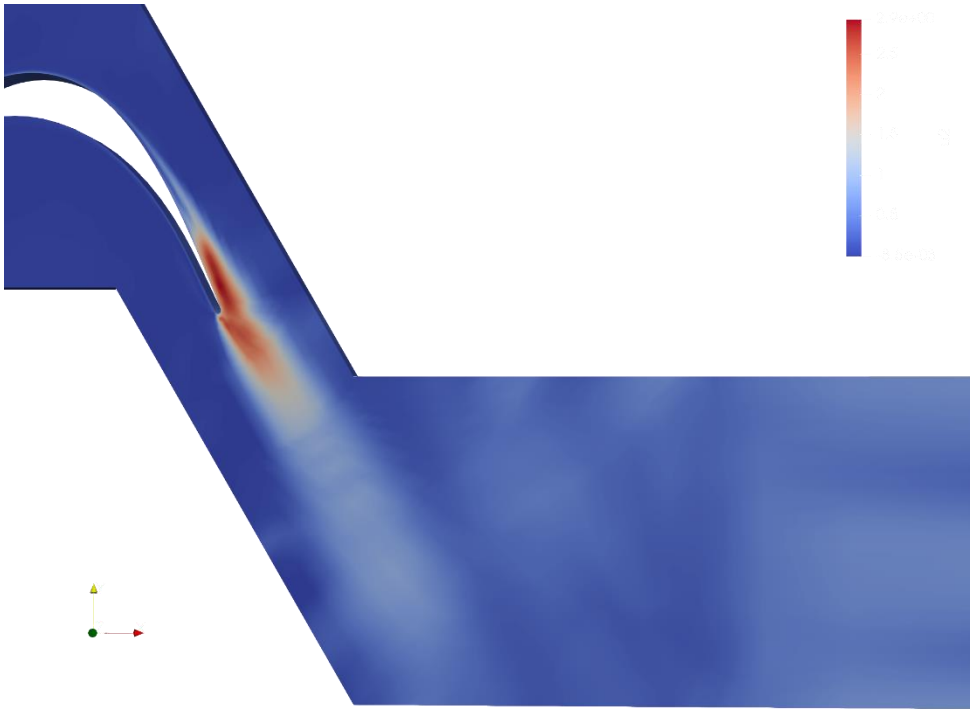

Figure 16: Raw Error Values by Larsson Average Indicator Machine Learning Model

The Reynold stress plots serve as another form of verification that the model has learned the pattern of the mesh adaptation formulas. After the base mesh is refined using the machine learning model the Reynold stress in the turbulent wake of the airfoil obtain very similar appearance as the DNS solution.

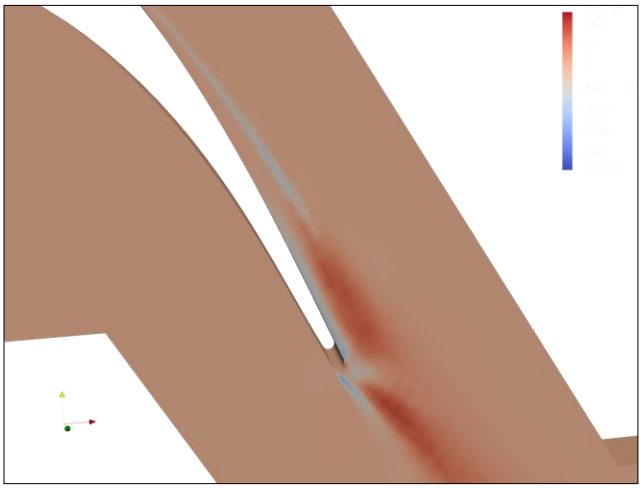

Figure 17: Reynold Stress Plot of the Base Mesh for the T106-C

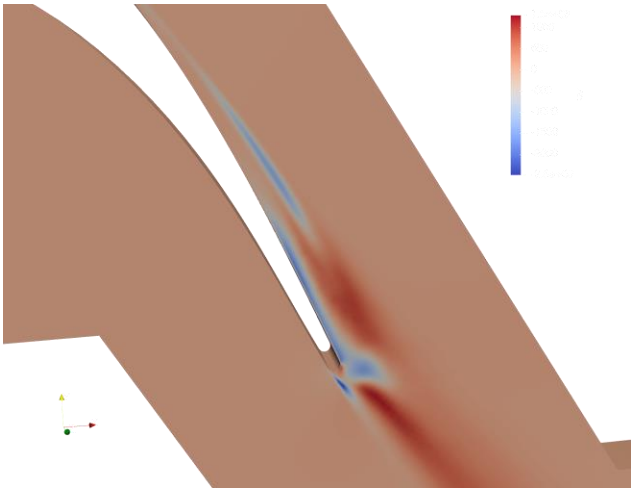

Figure 18: Reynold Stress Plot for the Refined Mesh Predicted by the Machine Learning Model

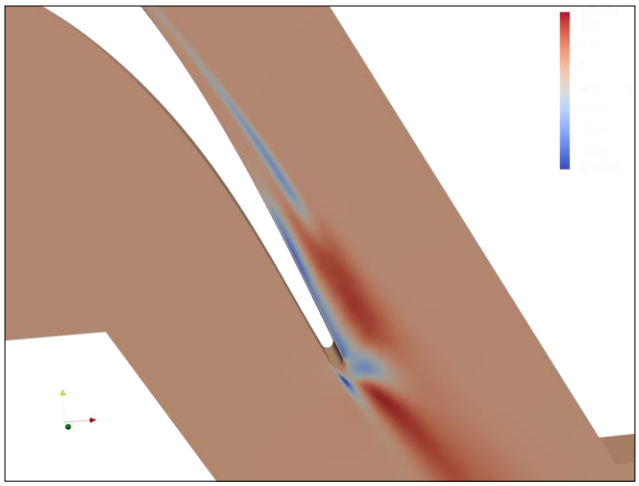

Figure 19: Reynold Stress Plot for the DNS Solution

The next set of results are for the second refinement level still using the Larsson average indicator. This refinement is the next mesh that the formulas or machine learning model produces after the first refined mesh. Here there are some differences between the formulas and the model. The model seems to not want to adapt as many cells near the outlet boundary condition and instead focuses on the cells near the airfoils surface. In terms of the turbulent wake both the formulas and the model seem to adapt similar cells. Figure 25 shows that the machine learning model adapts cells near the tip of the airfoils surface much more than the formulas chose to.

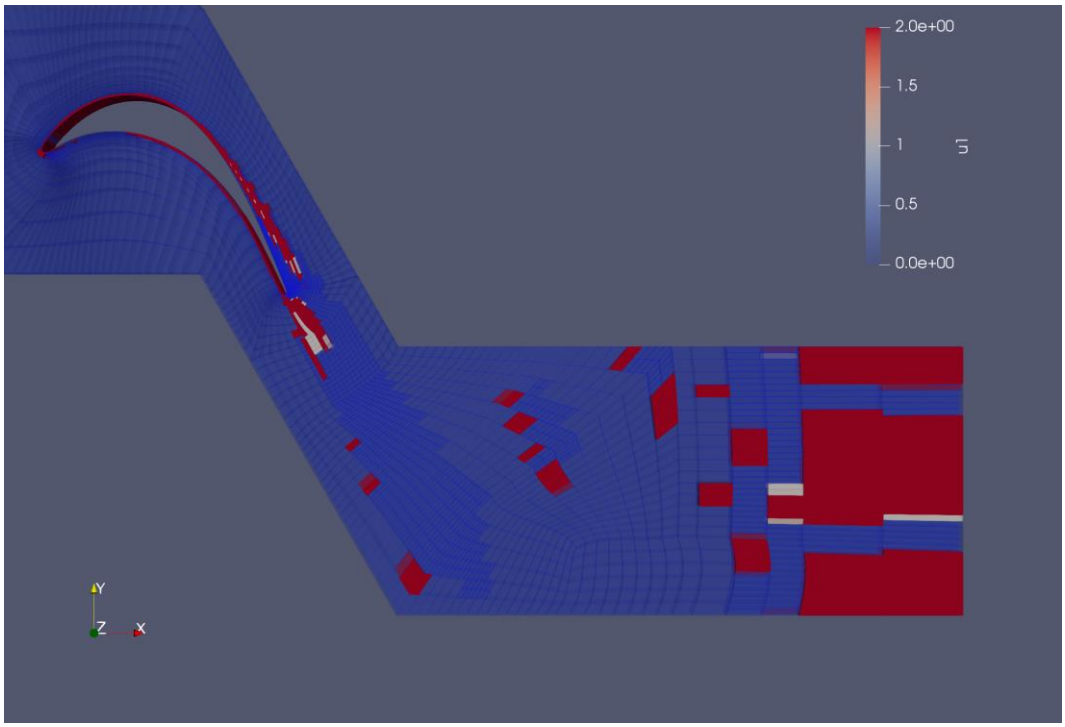

Figure 20: Cells Refined by Larsson Average Indicator Formulas, Refinement 2

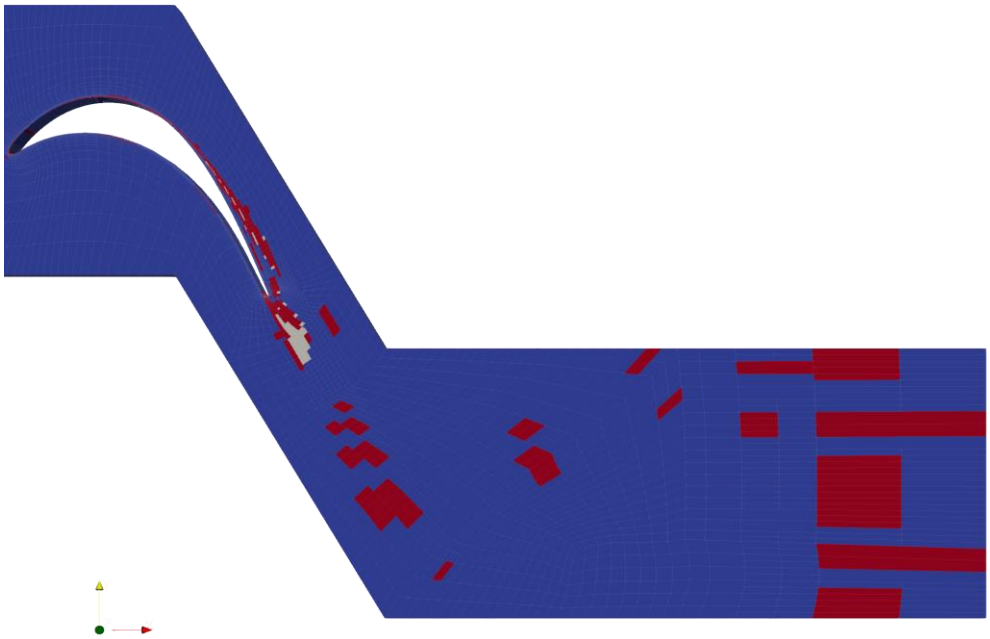

Figure 21: Cells Refined by Larsson Average Indicator Machine Learning Model, Refinement 2

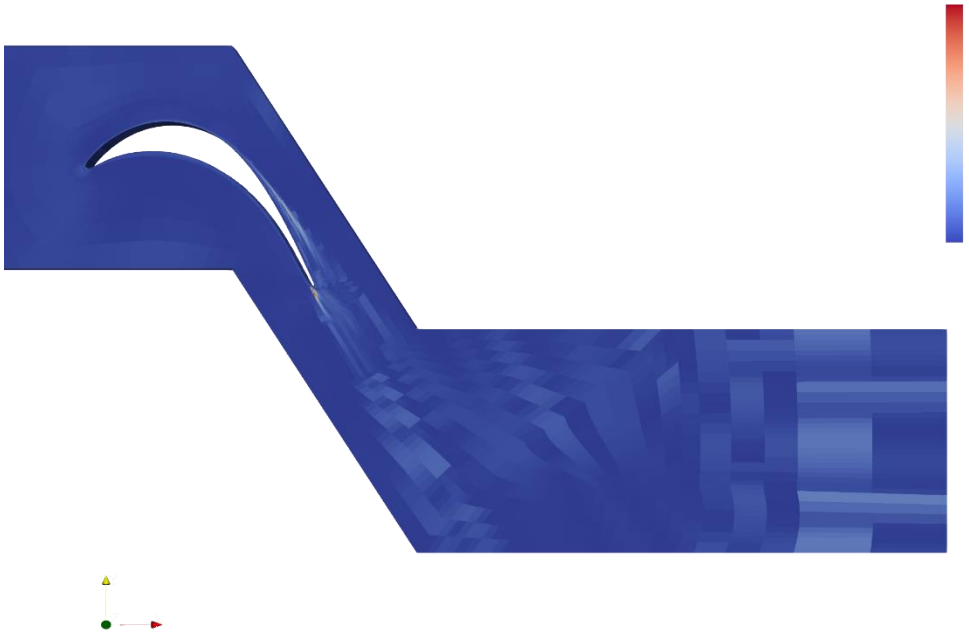

Figure 22: Raw Error Values by Larsson Average Indicator Formulas, Refinement 2

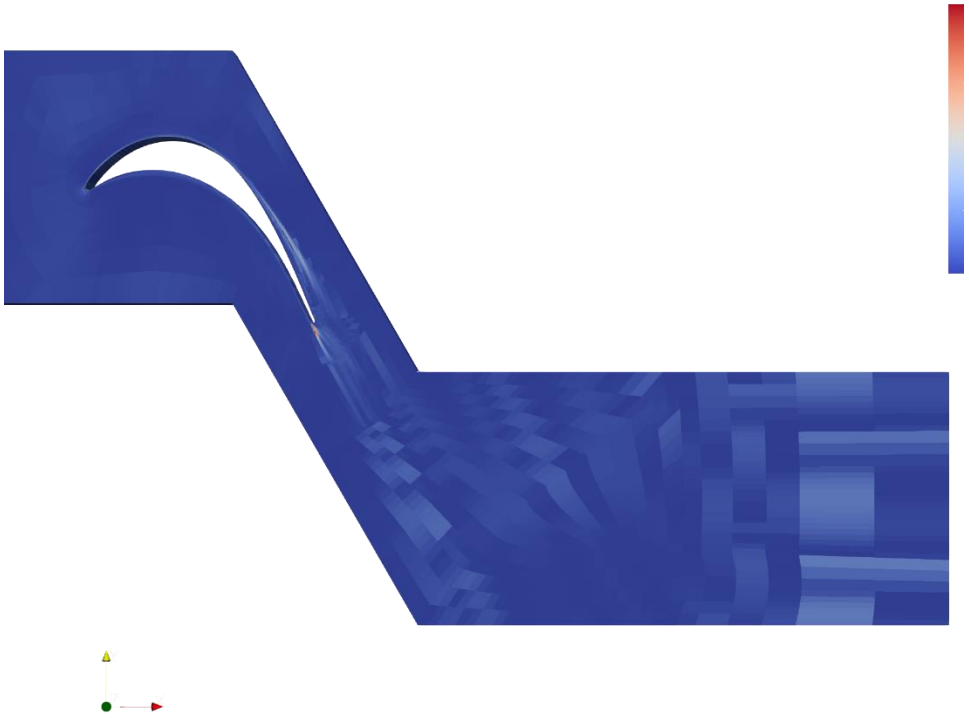

Figure 23: Raw Error Values by Larsson Average Indicator Machine Learning Model, Refinement 2

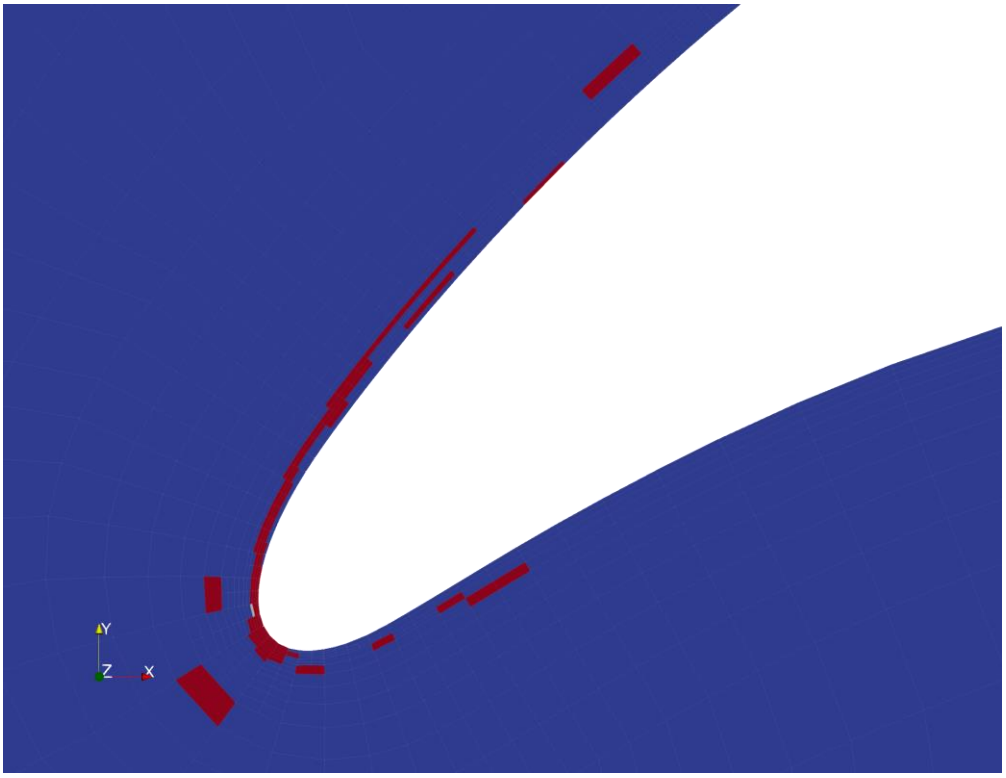

Figure 24: Close-up of the Cells Chosen to be Refined Near the Tip of the Airfoil by Larsson Average Formulas

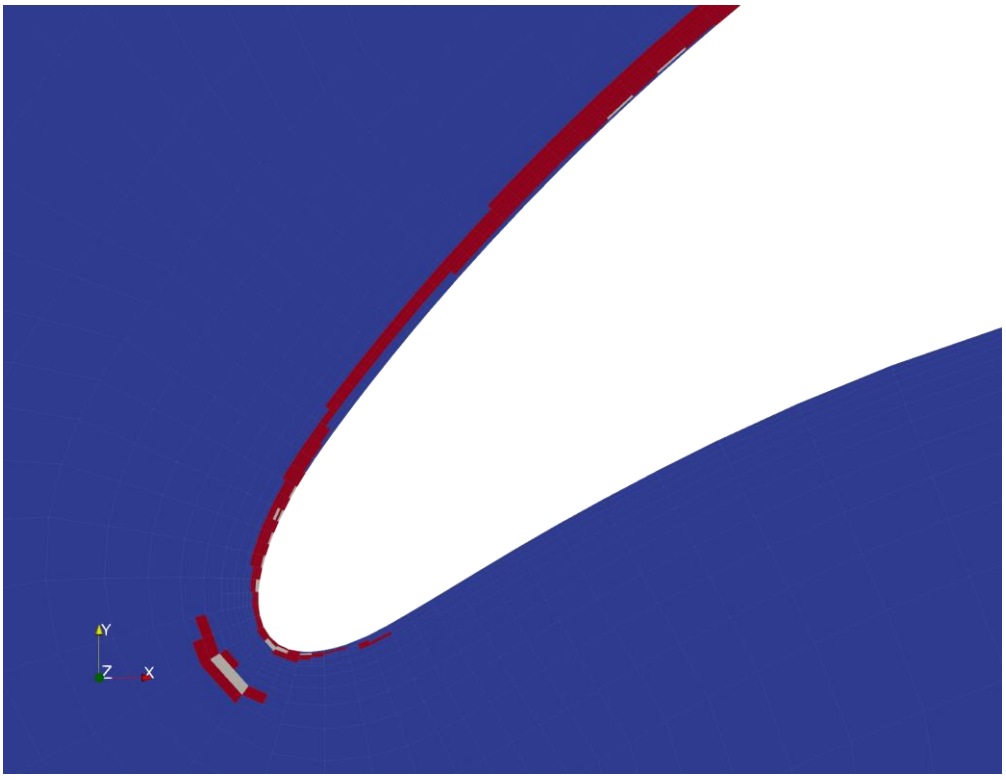

Figure 25: Close-up of the Cells Chosen to be Refined Near the Tip of the Airfoil by Machine Learning Model

4.2 Larsson & Toosi Based Indicator

The Larsson and Toosi based model show great converge with the training loss reaching 2.00e-3 and the validation loss reaching 3.60e-3. Though the model does not reach the same levels of convergence as the Larsson average model, the loss reduction and overlap of training and validation loss still show that the model was trained to an acceptable accuracy. This is better illustrated in Figure 27 as the plot shows some outliers from the diagonal line.

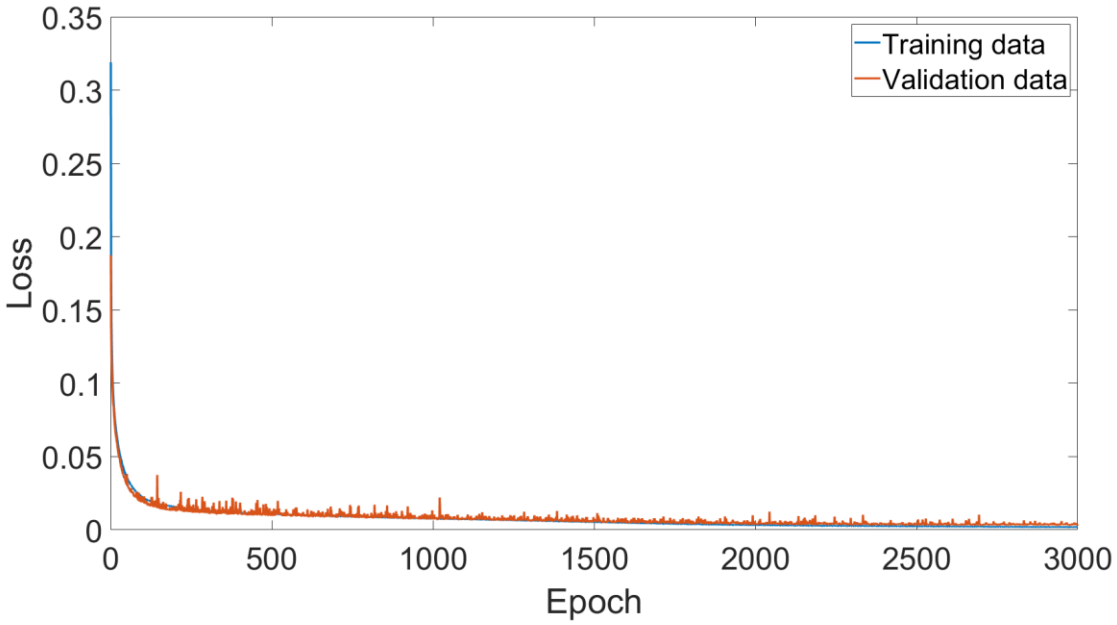

Figure 26: Loss vs Epoch for Larsson & Toosi Indicator Model

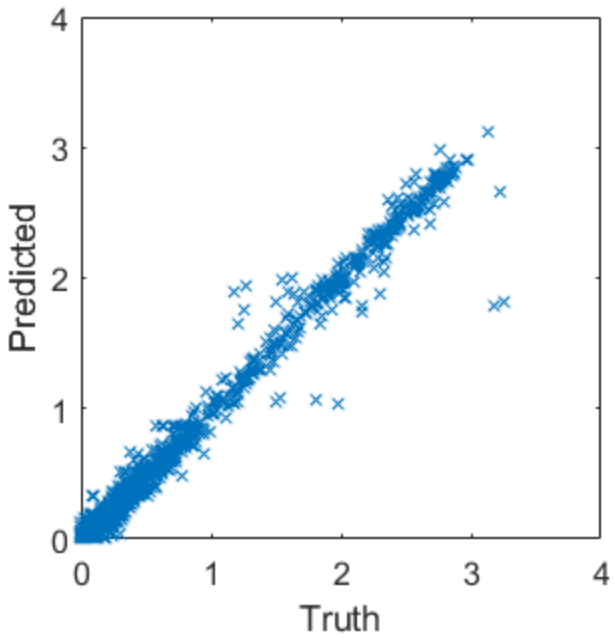

Figure 27: Predicted vs Truth for Larsson & Toosi Indicator Test Data

The difference between the formulas and the machine learning model in the Larsson and Toosi indicator is again mostly seen around the outlet boundary condition. In general, the turbulent wake cells are all chosen to adapt in both cases. With the machine learning model, it does seem to put more importance on the cells near the airfoil surface. In this case, it seems to be more beneficial as comparing to the DNS solution it captures an important structure not captured by the formulas. This structure is circled in Figure 29.

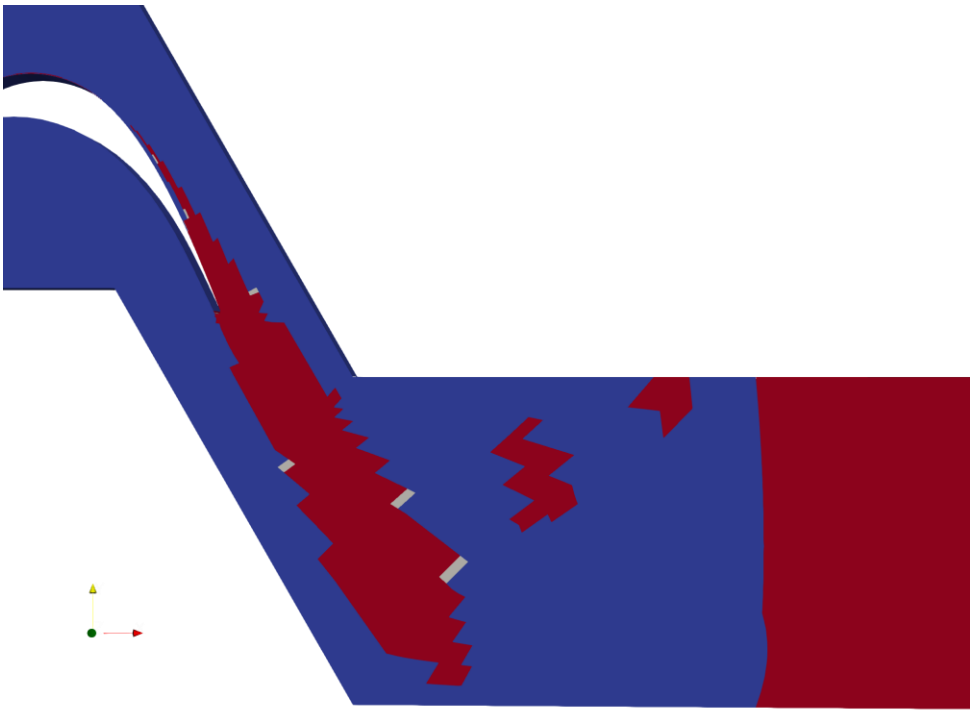

Figure 28: Cells Refined by Larsson & Toosi Indicator Formulas

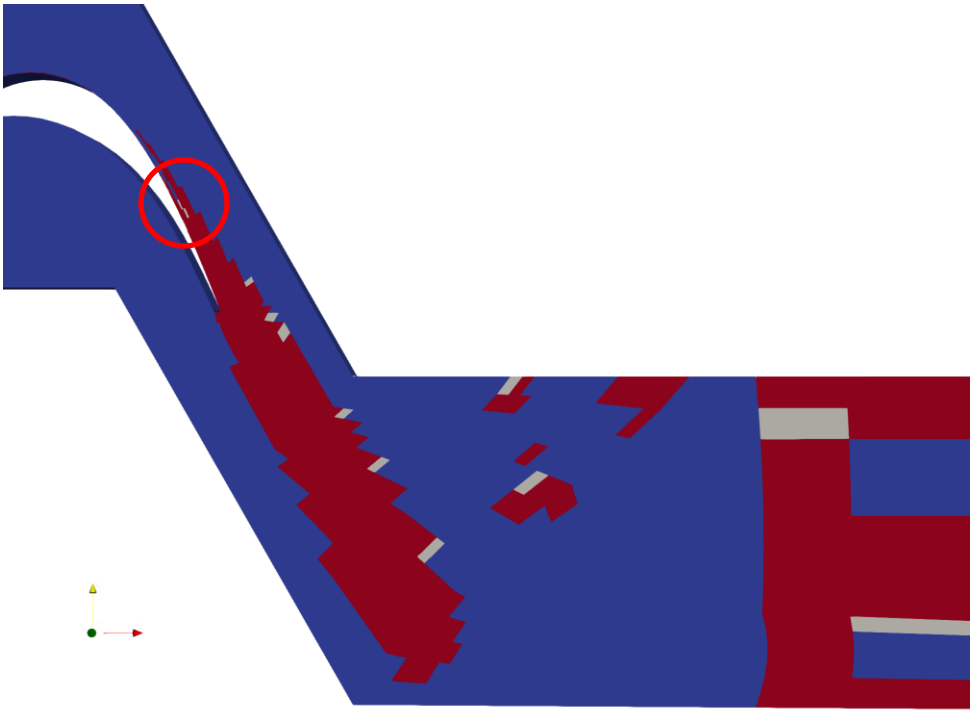

Figure 29: Cells Refined by Larsson & Toosi Indicator Machine Learning Model

5 Conclusions

To conclude, it has been shown that neural networks possess the ability to learn the complex adaptation patterns involved in mesh adaptation criteria to a high accuracy. This was shown by the reduction in training and validation loss reaching five order of magnitude for the Larsson average model and three orders of magnitude for the Larsson and Toosi model. By training the two different models based on either the unsteady flow field or the averaged flow field, it is shown that machine learning can be applied to mesh adaptation to lower the computational cost by reducing the complex mesh adaptation indicators into simple matrix multiplications. All the results show that the models predicted at a high accuracy when comparing with both the meshes obtained using the error indicators and the available DNS solution. In the important areas such as the turbulent wake and airfoil surface, the cells chosen for adaptation from both machine learning models closely resembled the cells chosen for adaptation from the error indicators. In the case of the Larsson and Toosi model, it may have even picked up on an important turbulent structure that the error indicators did not.

To continue the research shown in this paper, an extension could be made to see how the results will vary depending on if another element type other than hexahedrons were used for the training data mesh or if a mixed mesh would produce similar results. Another possible avenue of research would be to see how the same models trained in this research would refine the mesh of a problem with altered flow conditions. If possible, a model that could refine a mesh in a similar fashion to using the error indicators for any geometry would be the ideal case, though this most likely would require an unrealistically large data set.

6 References

Abbà A., Recanati A., Tugnoli M., Bonaventura L. "Dynamical p−adaptivity for LES of compressible flows in a high order DG framework,." *Journal of Computational Physics* (2020).

Alwin Hopf, Kai Velten, William Lubitz. "Simulation of airflow within horticulture high-tunnel greenhouses using open-source CFD software." *Hochschule Geisenheim University* (2018).

Anders Helgeland, B. Anders Pettersson Reif, Oyvind Andreassen, Carl Erik Wasberg. "Visualization of Vorticity and Vortices in Wall-Bounded Turbulent Flows." *IEEE* (2007).

Anthony F Molland, Stephen R Turnock. *Marine Rudders and Control Surfaces*. 2007.

B. Cockburn, GE Karniadakis, C. Shu. "Discontinous Galerkin Methods. Theory, computation and Applications." Berlin, 2000.

Bassi F., Rebay S. "A High Order Discontinous Galerkin Method for Compressible Turbulent Flows." *Discontinous Galerkin Methods: Theory, Compuation, and Application*. Springer, 2000.

Benard P., Balarac G., Moureau V., Dobrzynski C., Lartigue G., D'Angelo Y. "Mesh adaptation for large-eddy simulations in complex geometries." *International Journal for Numerical Methods in Fluids* (2015).

Brownlee, Jason. *https://machinelearningmastery.com/*. 2020. August 2021.

Daviller G., Brebion M., Xavier P., Staffelbach G., Poinsot T. "A mesh adaptation strategy to predict pressure losses in LES of swirled flows." *Flow, Turbulence, and Combustion* (2017).

Dwight, R. "Heuristic a posteriori estimation of error due to dissipation in finite volume schemes and application to mesh adaptation." *Journal of Computational Physics* (2008).

El-Banbi Ahmed, El-Maraghi Ahmed. *PVT Property Correlations*. 2018.

Franck, N. "Unsteady flows modeling and computation." (2007).

Gassner G., Dumnser M., Hindenlang F., Munz C.D. "Explicit one-step time discretization for discontinuous galerkin and finite volume schemes based on local predictors." *Journal of Computational Physics* (2011).

Huynh, HT. "A Flux Reconstruction Approach to High-Order Schemes Including Discontinous Galerkin Methods." *AIAA* (2007).

J. Gou, X. Su, X. Yuan. "Adaptive mesh refinement method-based large eddy simulation for the flow over circular cylinder at ReD = 3900." *International Journal of Computational Fluid Dynamics* (2018).

J. Peter, M. Nguyen-Dinh, P. Trontin. "Goal oriented mesh adaptation using total derivative of aerodynamic functions with respect to mesh coordinates - with applications to euler flows." *Computers & Fluids* (2012).

Jiachen Yang, Tarik Dzanic, Brenden Petersen, Jun Kudo, Ketan Mittal, Vladimir Tomov. "Reinforcement Learning for Adaptive Mesh Refinement." *Cornell University* (2021).

Jorge D. Rios, Alma Y. Alanis, Nancy Arana-Daniel, Carlos Lopez-Franco. *Neural Networks Modeling and Control*. 2020.

K. Matsuno, N. Satofuka, P. Fox, A. Ecer, J. Periaux. *Parallel Computational Fluid Dynamics*. 2003.

KrzysztofJ. Fidkowski, Guodong Chen. "Metric-based, goal-oriented mesh adaptation using machine learning." *Journal of Computational Physics* (2020).

Kwan-Liu Ma, J. Van Rosendale, W. Vermeer. "3D shock wave visualization on unstructured grids." *IEEE* (1996).

Levy D., Zickuhr T., Vassberg J., Agrawal S., Wahls R., Pirzadeh S., Hemsch M. "Data Summary from the First AIAA Computational Fluid Dynamics Drag Prediction Workshop." *Journal of Aircraft* (2003).

Liu Y., Vinokur M., Wang Z.J. "Discontinous Spectral Difference Method for Conservation Laws on Unstructured Grids." *Journal of Computational Physics* (2006).

Nemec M., Aftosmis M.J. "Adjoint error estimation and adaptative refinement for embedded-boundary cartesian meshes." *AIAA* (2007).

nvidia. *SuperVize Me: What's the Difference Between Supervised, Unsupervised, Semi-Supervised and Reinforcement Learning?* 2018.

O.Antepara, O.Lehmkuhl, R.Borrell, J.Chiva, A.Olivaa. "Parallel adaptive mesh refinement for large-eddy simulations of turbulent flows." *Computers & Fluids* (2015).

Odier N., Thacker A., Harnieh M., Staffelbach G., Gicquel L., Duchaine F., García Rosa N., Müller J. "A mesh adaptation strategy for complex wall-modeled turbomachinery LES." *Computers & Fluids* (2021).

P.C. Cui, Y.Q. Deng, J. Tang. "Adjoint equations-based grid adaptation and error correction." *AAAS* (2016).

P.J. Frey, F. Alauzet. "Anisotropic mesh adaptation for CFD computations." *Comput. Methods Appl. Mech. Engrg.* (2005).

Park M.A., Kleb B., Anderson W.K., Wood S.L., Balan A., Zhou B.Y., Gauger N.R. "Exploring Unstructured Mesh Adaptation for Hybrid Reynolds-Averaged Navier–Stokes/Large Eddy Simulation." *AIAA* (2020).

Park, Michael Andrew. "Adjoint-Based, Three-Dimensional Error Prediction and Grid Adaptation." *AIAA* (2012).

R. Hu, Q. Wang, P. Blonigan. "Least Squares Shadowing sensitivity analysis of chaotic limit cycle oscillations." *Volume 267* (2014).

R. Li, T. Tang, P. Zhang. "Moving mesh methods in Multiple dimensions based on harmonic maps." *Journal of Computational Physics* (2001).

Ramon, Quiza, J. Paulo Davim. *Computational Methods and Optimization*. 2011.

Roy, C.J. "Strategies for Driving Mesh Adaptation in CFD." *AIAA* (2009).

Roy, Christopher. "Strategies for Driving Mesh Adaptation in CFD (Invited)." *AIAA* (2009).

S.Balsara, Dinshaw. "Divergence-Free Adaptive Mesh Refinement for Magnetohydrodynamics." *Journal of Computational Physics* (2001).

S.T. Bose, G.I. Park. "Wall-modeled large-eddy simulation for complex turbulent flows." *Annual Review of Fluid Mechanics* (2018).

Shu, C. "Total-Variation-Diminishing Time Discretizations ." *SIAM Journal on Scientific and Statistical Computing* (1988).

Suavanan Lakshmanan, B. K. Soni, K. Balasubramaniam. "r-ADAPTATION IN FINITE ELEMENT MODELLING OF ELASTIC SOLIDS." *Department of Aerospace Engineering and Mechanics, Mississippi State University* (1995).

T. Haga, H. Gao, Z.J. Wang. "A High-Order Unifying Discontinous Formulation for the Navier-Stokes Equations on 3D Mixed Grids." *Math. Model. Nat. Phenom* (2011).

T. Shih, Y. Qin. "A Method for Estimating Grid-Induced Errors in Finite-Difference and Finite-Volume Methods." *AIAA* (2003).

Tingfan WU, Xuejun LIU, Wei AN, Zenghui HUANG, Hongqiang LYU. "A mesh optimization method using machine learning5technique and variational mesh adaptation." *Chinese Journal of Aeronautics* (2020).

Toosi Siavash, Larsson Johan. "ERROR ESTIMATION, GRID SELECTION AND CONVERGENCE VERIFICATION IN LARGE EDDY SIMULATION." *University of Maryland* (2019).

W. Cao, W. Huang, R.D. Russell. "An r-adaptive finite element method based upon moving mesh PDEs." *Journal of Computational Physics* (1999).

W.T. Jones, E.J. Nielsen, M.A. Park. "Validation of 3D adjoint based error estimation and mesh adaptation for sonic boom reduction." *AIAA* (2006).

Wang Z.J., Gao H. "A Unifying Lifting Collocation Penalty Formulation Including the Discontinous Galerkin, Spectral Volume/Difference Methods for Conservation Laws on Mixed Grids." *Journal of Computational Physics* (2009).

Wang, Z.J. "Progress in High-Order Methods for Industrial Large Eddy Simulation." *Department of Aerospace Engineering, University of Kansas* (2021).

—. "Relative Performance of High-Order over 2nd Order Schemes for Benchmark LES Problems." *AIAA* (2021).

—. "Relative Performance of High-Order over 2nd Order Schemes for Benchmark LES Problems." *AIAA* (2021).

Witten I., Frank E., Hall M., Pal C. "Chapter 10 - Deep learning." *Data Mining*. 2017. 417- 466.

Yano M., Darmofal D. L. "An Optimization-Based Framework for Anisotropic Simplex Mesh Adaptation." *Journal of Computational Physics* (2012).

Yi Li, Antony Jameson, Yves Allaneau. "Continuous Adjoint Approach for Adaptive Mesh Refinement." *AIAA* (2011).