Turbulence Modelling with Deep Neural Networks

Jie Bao

A Thesis

In the Department

of

Mechanical, Industrial, and Aerospace Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of

Master of Applied Sciences (Mechanical Engineering) at

Concordia University

Montreal, Quebec, Canada

May 2023

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to verify that thesis prepared

By: **Jie Bao**

Entitled: **Turbulence Modelling with Deep Neural Networks**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Mechanical Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final Examining Committee:

_____ Chair

Dr. Ebenezer Ekow Essel

_____ Examiner

Dr. Lyes Kadem

_____ Examiner

Dr. Ebenezer Ekow Essel

_____ Supervisor

Dr. Brian Vermeire

Approved by: _____

Chair of Department or Graduate Program Director, Dr. Sivakumar Narayanswamy

_____ 2022 _____

Dean of Faculty, Dr. Mourad Debbabi

Abstract

Turbulence Modelling with Deep Neural Networks

Jie Bao

High-Fidelity computational fluid dynamics simulations, such as large eddy simulation and direct numerical simulation (DNS), are computationally expensive. For this reason, the Reynolds-averaged Navier-Stokes (RANS) equations are more popular for industrial applications. They are obtained by deriving time-averaged properties, removing the need to resolve small-scale unsteady turbulent fluctuations. However, RANS requires a closure model to relate unknown Reynolds stresses to the time-averaged velocities. Traditional turbulence closure models are prone to inaccuracies, particularly for separated and transitional flows, due to the chaotic and anisotropic nature of turbulence. In this study, a data-driven approach is used to model turbulence leveraging high-fidelity data from DNS. A canonical turbulent channel case is considered at three different Reynolds numbers and compared with reference results. A deep neural network is trained on these three datasets, and then used to predict two intermediate Reynolds numbers. Comparison with parallel DNS simulations demonstrates agreement from 70% to 98% R2 score for predictions of turbulent kinetic energy production and dissipation, depending on the chosen training and testing datasets. A second study is then performed on a NACA 0012 airfoil at a Reynolds number of 50,000 at angles of attack ranging from 4 to 12 degrees. A deep neural network was trained on a limited set of these angles of attack, and used to predict turbulent kinetic energy production and dissipation. Comparison with the testing dataset showed good agreement at lower angles of attack, with limited agreement at high angles of attack beyond stall. Based on these results, we can conclude that deep neural networks can be trained to accurately predict turbulent kinetic energy production and dissipation for wall-bounded turbulent flows, but there are some limitations for massively separated flows. Future work will focus on directly incorporating these results in a RANS turbulence modelling framework.

Acknowledgments

I would like to express my gratitude to my parents, for their support and sacrifice that have allowed me to pursue this degree. Without them this thesis would not have been possible.
I would like to thank my lab mates for their help in getting up to speed in the beginning, and for the great laughs.
Finally, I would like to acknowledge the guidance and patience from my supervisor Dr. Brian Vermeire. I am tremendously grateful for the knowledge and time he shared for my research.

Contents

List of Figures

List of Tables

List of Acronyms

AOA Angle Of Attack

ANN Artificial Neural Network

AI Artificial Intelligence

CCC Constitutive Closure Coefficients

CFD Computational Fluid Dynamics

DGM Discontinuous Galerkin Method

DOF Degrees Of Freedom

DNS Direct Numerical Simulation

DRL Deep Reinforcement Learning

FEM Finite-Element Simulation

FR Flux Reconstructions

GPU Graphical Processing Unit

ILES Implicit Large-Eddy Simulation

LES Large Eddy Simulation

LSTM Long Short Term Memory

MDO Multi-Disciplinary Optimization

MLP Multi Layer Perceptron

ML Machine Learning

MSE Mean Squared Error

NF Nondimentionalizing Factor

NN Neural Network

PINN Physics-informed Neural Network

RANS Reynolds-averaged Navier-Stokes

ReLU Rectified Linear Unit

SGD Stochastic Gradient Descent

SGS SubGrid-Scale

TCC Transport Closure Coefficients

TBNN Tensor Basis Neural Network

TKE Turbulent Kinetic Energy

Chapter 1

Introduction

1.1 Green Aviation

The demand for global air transportation has risen to 4.5 billion passengers in 2019 [**ICAO_2019**]. Despite a reduction in passengers during the COVID 19 pandemic, the number of air passengers is expected to grow to 5 billion by 2025 [**IATA_2024**]. Using conventional aircraft designs, this will lead to a significant increase in greenhouse gas emissions, and detrimental environment impacts. Hence, developing efficient aircraft technologies will play a critical role in the growth of aviation going forward. The International Air Transport Association has published a technology roadmap for environmental improvement of the aerospace industry. These innovations could significantly reduce the fuel burn and environmental footprint of next generation aircraft.

For instance, a hybrid-electric aircraft using a blended wing body configuration can reduce CO_2 emission by up to 40% compared to the existing fleet of aircraft in the same category [**CO2**]. To close these technological gaps, advanced aerodynamic tools, capable of accurately capturing the complexity of turbulent flows are required for advanced aircraft designs. Increasingly, this relies on expensive simulation campaigns to simulate components and complete aircraft in a wide range of operating configurations. In order to minimize the cost of performing a large number of simulations, a range of approximate solution methods have been developed such as the low-order steady state Reynolds-averaged Navier-Stokes (RANS). These allow data to be approximated via a set of simplified time averaged equations using turbulence models. These turbulence models can be considered as physics-based non-linear interpolars from the benchmark problems they are tuned with.

Outside of the aerospace industry, Machine Learning (ML) has been widely used for similar

data interpolation applications. The objective of this thesis is to explore the utility of ML as an interpolating model for turbulent fields in benchmark turbulent flow problems, including a turbulent channel and turbulent NACA 0012 airfoil. It is expected that this could provide a new framework for reduced order modelling of complex turbulent flows, and that it could be extended for future aircraft designs.

1.2 High-order Methods

For industrial aerospace application, Computational Fluid Dynamics (CFD) is used as a design tool to complement the wind tunnel results and analytical methods. With increasing compute performance, RANS solvers are able to simulate more complex flows. However, the inherent inability to accurately convect and preserve the strength of vortical flow fields will limit their applications [**Ekaterinaris**]. The adoption of high-order schemes in combination with more accurate solution approaches, such as Direct Numerical Simulation (DNS) and Large Eddy Simulation (LES), would allow for coarser meshes with fewer Degrees Of Freedom (DOF) to be used for scale resolving simulations. Resulting in more cost-effective computation time for the same, or better, accuracy. In chapter 2, the Flux Reconstruction framework which uses a unstructured high-order element-wise Finite-Element Simulation (FEM) numerical method is presented in greater detail.

1.3 Artificial Intelligence

Artificial Intelligence (AI) can be described as the study of "intelligent agents"; such as a device making a decision that is obtained from reasoning after an observation made on its surrounding [**Norvig**]. Living in the information age, we benefit from many automated daily tasks with the help of so-called "narrow AI". These systems are on the same level or even surpassing human performance in a specific set of activities. The latter can be as simple as a game of chess [**Kasparov**]. However, modern AI still lacks the cognitive capabilities of humans, meaning it is incapable of generalizing to other similar types of tasks. The quintessential goal in the multidisciplinary field of AI is to develop a "strong AI" with the same general intelligence found in human beings [**AGI**]. However, the possibility to create strong AI is a controversial subject, and there are arguments on whether such a system is possible [**Dreyfus**, **Ragnar**].

The primary interest for this thesis concerns the ML subfield of AI. ML algorithms exists in

many forms, but all learn from the data they are given. In other words, a ML model will only be as good as the set of features that it is fed during training. Deep learning is a sub-branch of ML where the computer is able to express complex patterns using simpler representations from the visible layer [**Goodfellow**]. The first deep learning attempt started in 1943 with the development of connected neurons treated as logical gates. The result was the Artificial Neural Network (ANN), which was inspired by the nervous system of the human body, hence the name ANN [**McCulloch**]. It is also referred to as a Neural Network (NN) for short. A resurgence of ML in the 21st century was possible with breakthrough strategies such as greedy layer-wise pretraining [**Hinton**] and with increasingly more powerful computational resources allowing for more computationally expensive experiments. This new wave of popularity not only brought focus on new unsupervised and reinforcement learning techniques, but also to supervised learning algorithms that have existed since the 1980s.

Supervised Learning

Supervised learning is where the computer is provided with a set of training data and corresponding labels. If the label is *discrete*, such as to accurately identify the facial identity of a person [**Taigman**], then the learning process is called classification. Otherwise, the labels are *continuous*, such as the prediction of scaled sound pressure level generated by an airfoil [**decibels**], then the task is a regression. Surprisingly, the same learning algorithms from the 1980s are still used today for complex tasks [**Goodfellow**]. What has changed is the increased dataset size allowing them to train using more comprehensive and diverse data. As a general rule of thumb, for supervised learning it is recommended to have at least 5,000 labeled examples per category. To exceed human performance the training dataset needs to contain at least 10 million examples [**Goodfellow**].

Semi-Supervised Learning

Learning algorithms training on a dataset missing a portion of the target values is referred to as semi-supervised learning. One such field is Deep Reinforcement Learning (DRL) where an agent tries to maximize the reward it receives from a dynamic environment. The action taken by the agent in the system will inform the algorithm if a positive or a negative feedback (bounded real number) should be awarded. In turn, the action taken by the agent will change the environment (state) for the next time step. This also means DRL cannot be trained on a typical static dataset. For each action, a different state is generated by the interactive environment. However, what

works well in DRL networks is the sparse award system, meaning it does not require a feedback from each action of the agent. [**Sutton1998**]. It is proven that learning a long-term strategy or a set of moves works quite well for DRL algorithms. That is why most of the progress in DRL research is applied with games [**ATARI**].

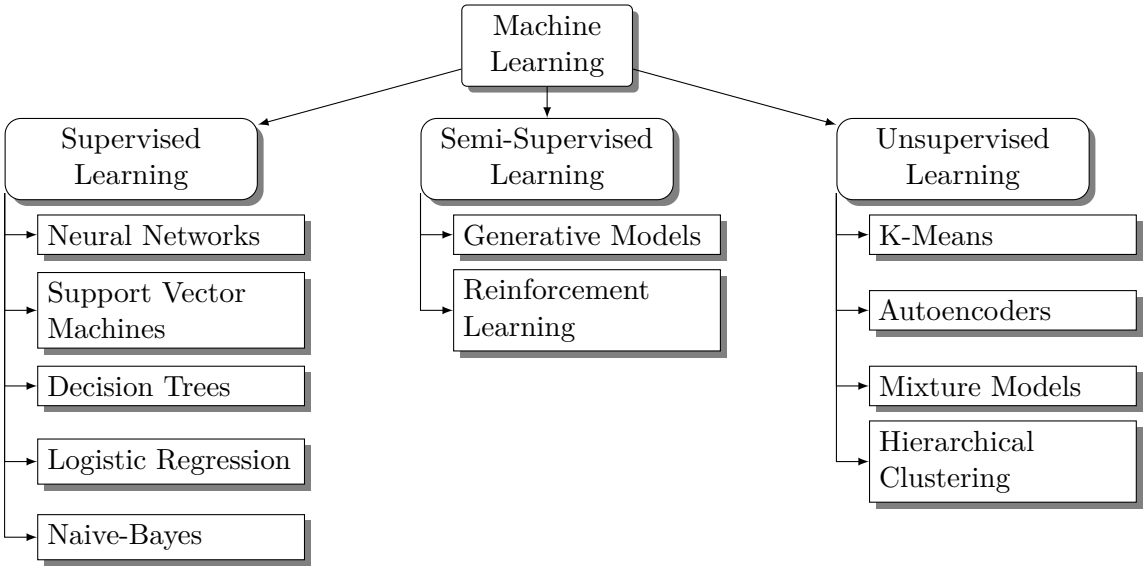

Figure 1.1. Summary of machine learning techniques

Unsupervised Learning

Unsupervised learning, refers to the absence of labeled targets in the given dataset. An unsupervised learning algorithm has the task to extract patterns, which can be done by clustering the dataset into different groups. Some common techniques are the k-means, mixture models, and hierarchical clustering [**top10**].

1.3.1 Machine Learning in Aerospace

Aircraft design is an inherent Multi-Disciplinary Optimization (MDO) process. To deliver an efficient modern aircraft that satisfies the requirements from several different disciplines. These disciplines include aerodynamics, structure, controls, to business cost analysis, and environmental impacts, amongst many others [**Raymer**, **MDO-Structure**]. This MDO framework lays the perfect groundwork for AI/ML application.

Machine learning has a wide range of capabilities limited by the volume of available data. However, aerospace applications typically produce and store large amounts of data. For instance,

an aircraft will generate thousands of performance and operations datasets throughout its life cycle. That data is generated from the design phase, the production, and all the way to the operational and maintenance phases. Aero Montreal published a report in 2019 [**AERO_MTL**] stating that Montreal is both one of the largest aerospace hubs and largest AI hubs in the world. Therefore, the aerospace industry in Montreal is poised to benefit from the collaboration those two sectors through cooperation. For instance, the strategic agreement between Bombardier, Scale AI and the Institute for Data Valorization (IVADO) to adopt preventive monitoring techniques using analytics models [**IVADO**, **SCALEAI-BBD**].

Example Applications in Industry

In the final stages of aircraft assembly, large gaps between high tolerance components is common. A time-consuming shim is custom made to satisfy the engineering nominal specifications. Researchers at the University of Washington have made progress using machine learning and sparse sensing for predicting shim gaps in aircraft assembly. They use robust principal component analysis to extract patterns in the gap measurements [**RPCA**]. Training on historical datasets provided by Boeing Aerospace, the accuracy of this model ranged from 96-99%. This proof of concept can improve assembly line efficiency and, over the years save billions of dollars [**shimming**, **sensing**]. Unscheduled maintenance on arrival is not unusual.

Unfortunately, disturbing an optimized flight schedule can have costly consequences. Therefore, research has focused on training neural networks to detect anomalies in data from flight sensors. Long Short Term Memory (LSTM) networks are particularly appealing due to their quality in learning from sequences and have been found to be useful in overcoming limitations from the past. Used as a means to identify events that reduce safety margins [**LSTM1**], and engine remaining useful life [**LSTM2**]. Convolution neural networks are also used in the fatigue crack diagnosis for aircraft structures [**CNN-Fatigue**]. Other research in the field of aerospace fluid mechanics involves reinforcement learning to control wake separation. Altering the aerodynamic flow field around a body by injecting energy or momentum is known as active flow control. Researchers have been able to train an agent through deep reinforcement learning to promote drag reduction by using small jets on the top and bottom of a 2D cylinder [**DLR-Garnier**]. This approach can also be an alternative to grid search for shape optimization problems [**DLR-Rabault**].

It is important to mention that machine learning will not replace theoretical and experimental methods. Rather it tries to leverage optimization directed on data to enhance the established

techniques and allow new discoveries [**paradigm**]. To improve the desired outcome, we may also embed physical learning process within the neural network's architecture.

1.3.2 Physics-based Neural Networks

Recent advances in Graphical Processing Unit (GPU) has equipped researchers with the capability to tackle even more computationally challenging problems [**VERMEIRE**]. Along with the investment on open-source AI software by many industry leaders [**Tensorflow**]. Now more than ever, non-experts have the opportunity to build neural networks for aerospace applications. Using increasingly better data collection and advances in data quality. A growing field known as "Data Science" capitalizes on the vast amount of data to make predictions with the goal to augment scientific research capabilities [**50years**].

However, even with so much data we cannot blindly apply a machine learning technique and expect accurate results. Instead, we can use existing model structures and integrate prior knowledge from a trained machine learning model. This is a data-driven approach and is currently applied in turbulence modeling research in the field of CFD. For instance, there are efforts for improving the LES by developing a neural networks to model the SubGrid-Scale (SGS) stress tensor (6 NN for all components). Training data are provided by DNS of a turbulent channel flow. Then a feed-forward neural network is used to establish the functional relationship between the grid-scale and the SGS. The result shows the NN models were not far from the DNS but only when the filter size was small: $\overline{\Delta}^{+} \lesssim 20$ [**Hattori**].

A more tractable turbulence modeling technique but less accurate is the RANS approach. Machine learning techniques have previously been used to model the RANS closure coefficients. From the seminal work of Ling *et al.* [**preLing**], a random forest regression model is used to predict the coefficients of the Reynolds anisotropy tensor expansion [**pope_1975**]. This technique presented poor result due to the difficulty to ensure Galilean invariance. Later Ling *et al.* used a NN architecture also known as a Tensor Basis Neural Network (TBNN) [**Ling**, **Ling2**, **Ling3**]. The TBNN differs from other models as it has embedded physical constraints thus ensuring Galilean and rotational invariance. Researchers from Texas A&M have advanced the TBNN introducing a closed loop framework [**Texas**]. The Constitutive Closure Coefficients (CCC) are extracted from the neural network to modify the Transport Closure Coefficients (TCC). The process stops once the CCC has converged. The approach proved to be better than the predictive accuracy of open loop framework.

CFD for aerospace application often involves the study of wing design at a high Reynolds number. However, the strong adverse pressure induces an anisotropy around the airfoil. The Spalart-Allmaras model purpose is to overcome the latter challenge for wall-bounded flows. The one equation model works by solving a viscosity-like variable $\tilde{\nu}$ and can be modified to account for compressible flows. Zhu *et al.* [**SAZhu**] used the Spalart-Allmaras data as training to build an eddy viscosity mapping function. The NN accuracy was validated and the approach proved to be more efficient than the original Spalart-Allmaras model due the absence of complex calculations.

Novati *et al.* used DRL to benefit the area of turbulence modeling. The authors of multi-agent reinforcement learning (MARL) demonstrated the potential of DRL on LES [**RLLES**]. The agents are dispersed in the simulation closure model and each agent performs a localized action based on information about the state of the local flow field. To estimate the unresolved subgrid-scale physics, the control policy enacted by the cooperating agents is to detect critical spatio-temporal patterns in the flow field. The encouraging result compares positively with state-of-the-art SGS modeling approaches. They also avoid the necessity to perform DNS during training through experiment measures. In turn used to train SGS models for LES. Their work focused solely on homogeneous and isotropic turbulence. Nevertheless, they opened a new research direction to consider DRL as a training algorithm for SGS closures.

An alternate approach to use TBNN as a means to obtain eddy viscosity still requires numerous iterations of data fitting for the CCC terms. The goal would be to bypass the need for a classical two equation model. Using the Turbulent Kinetic Energy (TKE), production and dissipation values we could infer the eddy viscosity directly. In this thesis, we take a first step in this direction by training a NN to infer the production and dissipation values.

1.4 Thesis Objectives

The goal of this thesis is to use open-source machine learning software to bypass the need for a RANS turbulence model. Training data will be generated using DNS with PyFR [**WITHERDEN20143028**]. Two three-dimensional simulation cases are considered; The turbulent channel case from Kim, Moin, and Moser (K.M.M) [**KMM**] and the turbulent flow around the NACA-0012 airfoil from Rodriguez *et al.* [**NACA0012**]. Favre-averaging is applied to consider compressibility effects in the flow. A deep neural network will be built to recognize the production and dissipation terms in the turbulent kinetic transport equation.

1.5 Thesis Outline

In Chapter 1, we introduced the nature of the work in this thesis and the overall state of artificial intelligence in the aerospace sector. Chapter 2 explains the fundamental theory behind most of the research, including fluid mechanics, high-order computational fluids and turbulence modeling techniques. The deep feedforward neural network theory is explained with the help of an example dataset in Chapter 3. Along with the general framework on how the ML is integrated into the CFD workflow. Results and discussion of the findings are located in Chapter 4. Finally, the conclusions and recommendations for future work are presented in Chapter 5. An excerpt of the end-to-end machine learning pipeline is located in the Appendix and also additional CFD validation figures.

Chapter 2

Theoretical Concepts

2.1 Governing Equations

The evolution of the physical properties of a fluid are described by a set of conservations laws. In this section, a brief description of a compressible fluid is presented using the Navier-Stokes equations. The detailed conservation law derivation can be found in [**davidson2004turbulence**].

Conservation of Mass

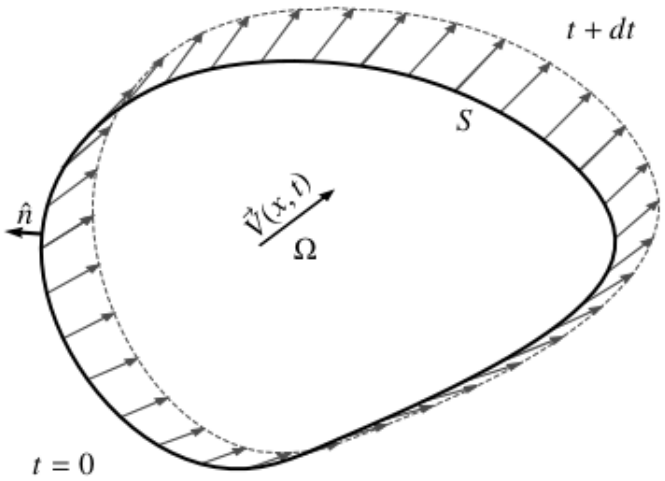

Figure 2.1. Schematic representation of Control Volume [**VermeireBook**]

Consider a volume of fluid, denoted as Ω, travelling across a medium of surface S. The mass passing through the region cannot disappear nor be created, and the volume flux with the surface is described by $\vec{f} = \vec{V} \cdot \hat{n}$. Using the conserved variable density, ρ. We can apply the general

conservation of mass on the Navier-Stokes equation,

$$\frac{\partial}{\partial t}\int_{\Omega}\rho\, d\vec{x} + \oint_{S}\rho\vec{V}\cdot\hat{n}\, dS = 0. \tag{2.1}$$

Applying Gauss's theorem, we obtain the divergence form of the convervation of mass

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\vec{V}) = 0. \tag{2.2}$$

Conservation of Momentum

According to Newton's second law, the time rate of change of the dot product between a mass and its velocity must be equal to all the forces acting on it.

$$\frac{\partial\rho}{\partial t}\int_{\Omega}\rho\vec{V}\, d\Omega + \oint_{S}\left(\rho\vec{V}(\vec{V}\hat{n} - \hat{n}\sigma)\right)\, dS = \int_{\Omega}\vec{F}\, d\Omega, \tag{2.3}$$

where the left hand side of Equation 2.3 is the explicit expression of the convective derivative. The stress tensor, σ, includes pressure and shear stresses on the surface, S. The right hand side contains the body forces. An important observation from Equation 2.3 is the non-linear quadratic term for $\vec{V}$. This non-linear convection, combined with entropy generation via diffusion, is responsible for the complex turbulent flows.
The divergence form is written using Gauss's theorem as

$$\frac{\rho\vec{V}}{\partial t} + \nabla\cdot(\rho\vec{V}\cdot\vec{V} - \sigma) = \vec{F}. \tag{2.4}$$

Conservation of Energy

In a closed system, the change in energy is equal to the rate of heat addition less the rate of work done by the system on its surroundings. The specific energy, E, is taken to have two parts, the kinetic and thermal energy.

$$E = e + \frac{\vec{V}\cdot\vec{V}}{2} \tag{2.5}$$

$$e = c_v T, \tag{2.6}$$

where e is the internal energy, c_v is the specific heat coefficient at constant volume, T is the temprature, and where $\vec{V}\cdot\vec{V}/2$ corresponds to the specific kinetic energy. By considering external

heat sources, $\vec{q}$, and work from viscous shear stresses and pressure, the conservation law for energy is

$$\frac{\partial}{\partial t}\int_{\Omega} \rho E \, d\Omega + \oint_{S} \cdot \left[\rho E\vec{V} - \rho\sigma\vec{V} + \rho\vec{q}\right] \cdot \hat{n} \, dS = \int_{\Omega} \vec{F} \cdot \vec{V} \, d\Omega, \tag{2.7}$$

and the divergence form is

$$\frac{\partial \rho E}{\partial t} + \nabla \cdot (\rho E\vec{V} - \rho\sigma\vec{V} + \rho\vec{q}) = \vec{F} \cdot \vec{V}. \tag{2.8}$$

2.2 The Flux Reconstruction Method

One-Dimensional Spatial Discretization

In 2007, Huynh introduced the Flux Reconstructions (FR) approach for solving the Navier-Stokes system of equations. The high-order method has demonstrated to have improved accuracy at lower computational cost than current industry-standard second-order schemes [**Huynh**]. By varying a single correction factor, the FR method can recover existing schemes such as the Discontinuous Galerkin Method (DGM). The FR approach will be explained using linear advection in a one dimensional domain. A general one-dimensional conversation law in divergence form is

$$\frac{\partial u}{\partial t} + \frac{\partial F}{\partial x} = 0, \tag{2.9}$$

where a conserved scalar $u = u(x,t)$, and the flux is $F = F(u)$.

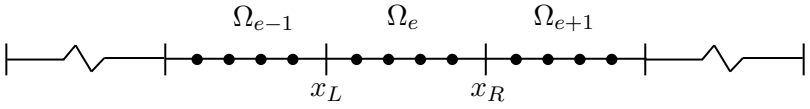

Figure 2.2. One-dimensional domain discretization

The computational domain, Ω, is split into N_e non-overlapping elements defined by $x \in [x_L, x_R]$ as shown in Figure 2.2. On each element, the solution and flux are represented by a polynomial of degree $N = N_p - 1$, where N_p is the number of nodal points. To handle elements of different sizes, we can use a standardized mapping transformation between the physical, Ω_e, and a reference space, $\xi \in [-1, 1]$,

$$\underbrace{x}_{physical\ coordinate} \longleftrightarrow \underbrace{\xi}_{reference\ coordinate} \tag{2.10}$$

$$\xi(x) = \frac{2}{h_e}\left(x - \frac{x_L + x_R}{2}\right), \tag{2.11}$$

where h_e refers to the physical element size. Note the Jacobian of this mapping is

$$\frac{\partial \xi}{\partial x} = \frac{2}{h_e} \rightarrow \frac{\partial x}{\partial \xi} = \frac{h_e}{2}. \tag{2.12}$$

The invertible transformation can be manipulated to output the spatial derivative from the chain rule. For instance the flux derivative

$$\frac{\partial F}{\partial x} = \frac{\partial F}{\partial \xi}\frac{\partial \xi}{\partial x}. \tag{2.13}$$

The solution is represented by nodal basis points and their corresponding nodal basis function. The FR is a type of finite-element method that allows discontinuous solutions between neighbouring elements. The Lagrange polynomials are used as nodal basis functions. Therefore, in every element, Ω_e, we have a local solution $u_e(x,t)$,

$$u_e^h(\xi,t) = \sum_{k=1}^{N_p} \underbrace{u_{e,k}(t)}_{nodal\ coeffi.} \underbrace{\phi_k(\xi)}_{nodal\ basis\ fct.}, \tag{2.14}$$

$$f_e^h(\xi,t) = \sum_{k=1}^{N_p} f_{e,k}(t)\phi_k(\xi), \tag{2.15}$$

where $u_e^h(x,t)$ and $f_e^h(x,t)$ are parts of a global piecewise continuous approximation of the solution on a single element. We then assume the true global solution $u(x,t)$ is approximated by the direct sum of all of the $u_e^h(x,t)$

$$u(x,t) \approx u_e^h(x,t) = \bigoplus_{e=1}^{N_e} u_e^h(x,t), \tag{2.16}$$

Additionally, ϕ_k is the kth nodal basis function in the reference space. The latter is governed by the Lagrange polynomials. In the case of one-dimension

$$\phi_k(\xi) = \prod_{m=1,\, m\neq k}^{K} \frac{\xi - \xi_m}{\xi_k - \xi_m} \tag{2.17}$$

These nodal basis function take the value of 1 at their respective solution point and zero at the other solution points.It has been observed that an equidistant distribution of the basis points could produce undesired oscillations for non-linear flux functions. This behaviour could be suppressed by clustering the points near both end of the element. As shown in the next page, the Gauss-Legendre points are preferred in this case.

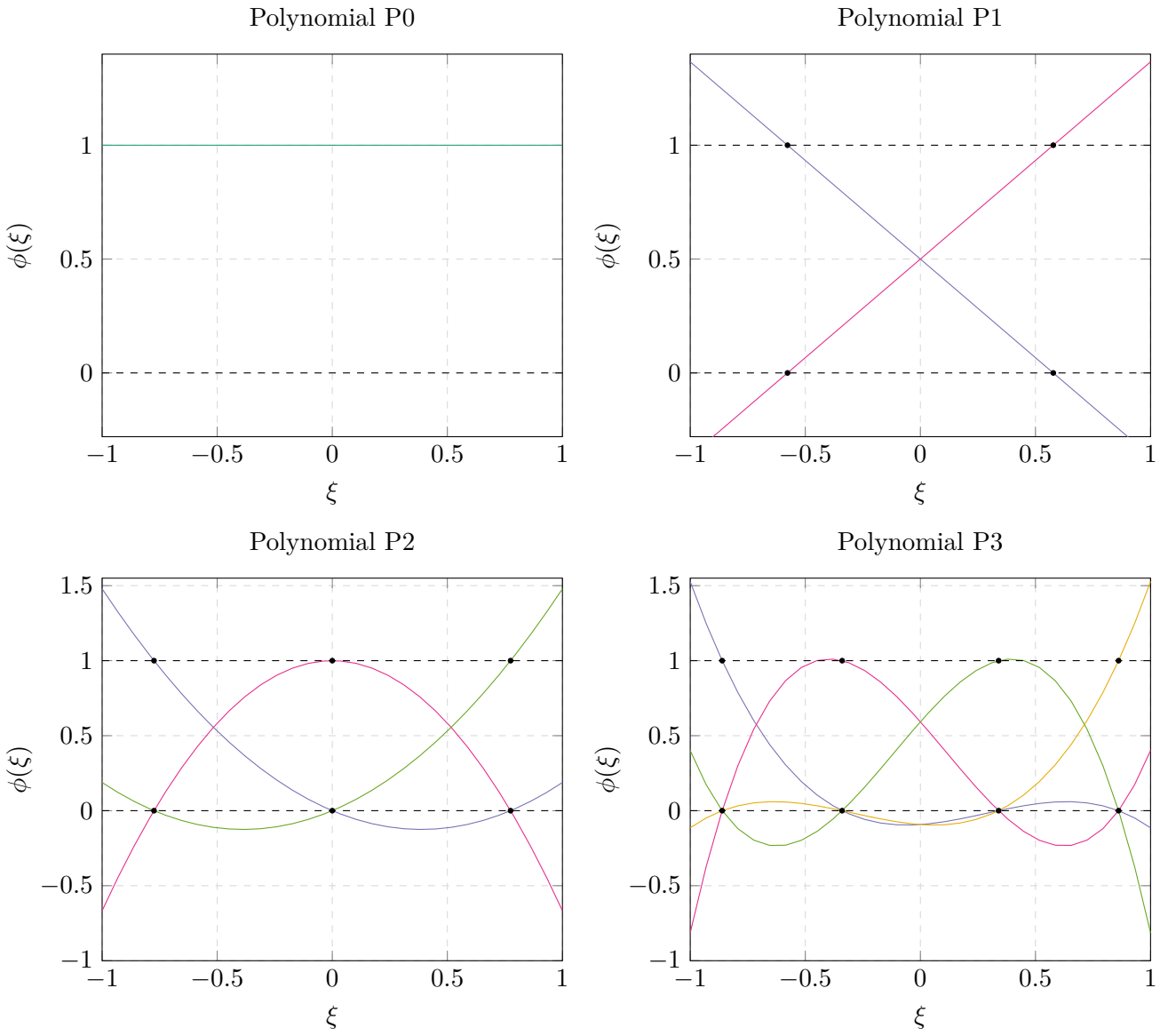

Figure 2.3. Nodal basis functions for 1D elements using P0 to P3

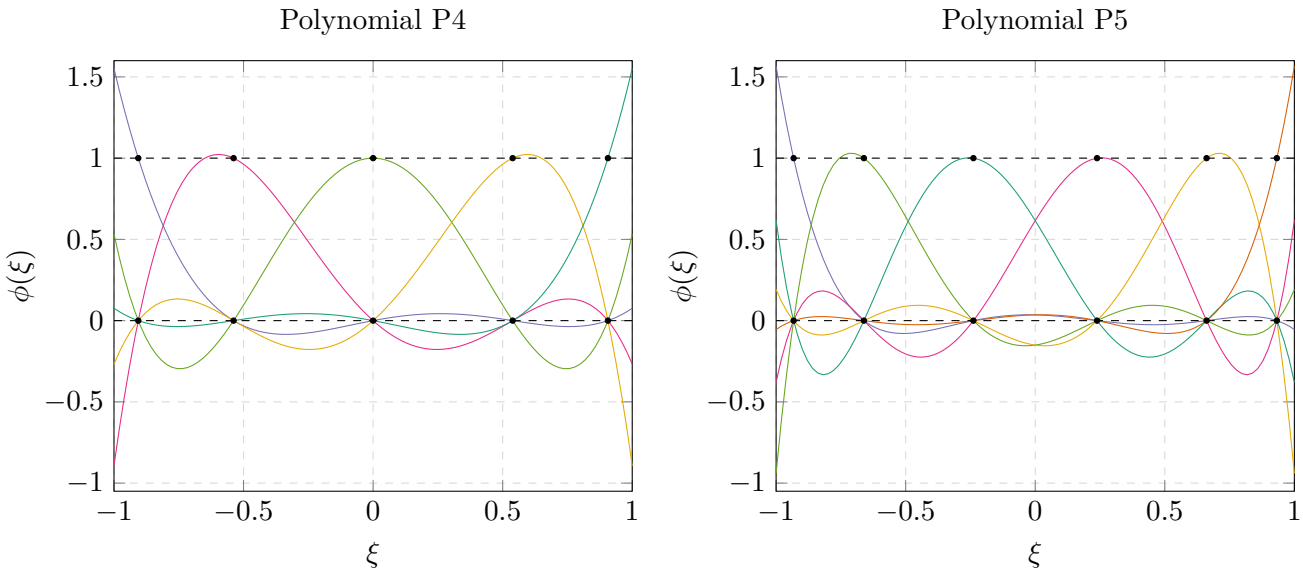

Figure 2.4. Nodal basis functions for 1D elements using P4 and P5

An alternate representation using orthonormal modal basis functions is

$$u_e^h(\xi, t) = \sum_{k=1}^{N_p} \tilde{u}_{e,k}(t)\tilde{l}_k(\xi), \tag{2.18}$$

Where $\tilde{u}_{e,k}$ are the k-th modal expansion coefficient. These are the well-known *Legendre polynomials*, $\tilde{l}$, as shown in figure 2.5.

The two forms are used for polynomial construction and can be translated from one to another using the *Vandermonde matrix*, V

$$u = V\tilde{u}. \tag{2.19}$$

Where $V_{i,j} = x_i^{j-1}$. The approximate solution should satisfy

$$\frac{\partial u_k^h}{\partial t} + \frac{\partial f_k^h}{\partial \xi} = 0. \tag{2.20}$$

However, we know the latter cannot be equal to the exact solution, resulting in a residual $R(\xi, t) \neq 0$.

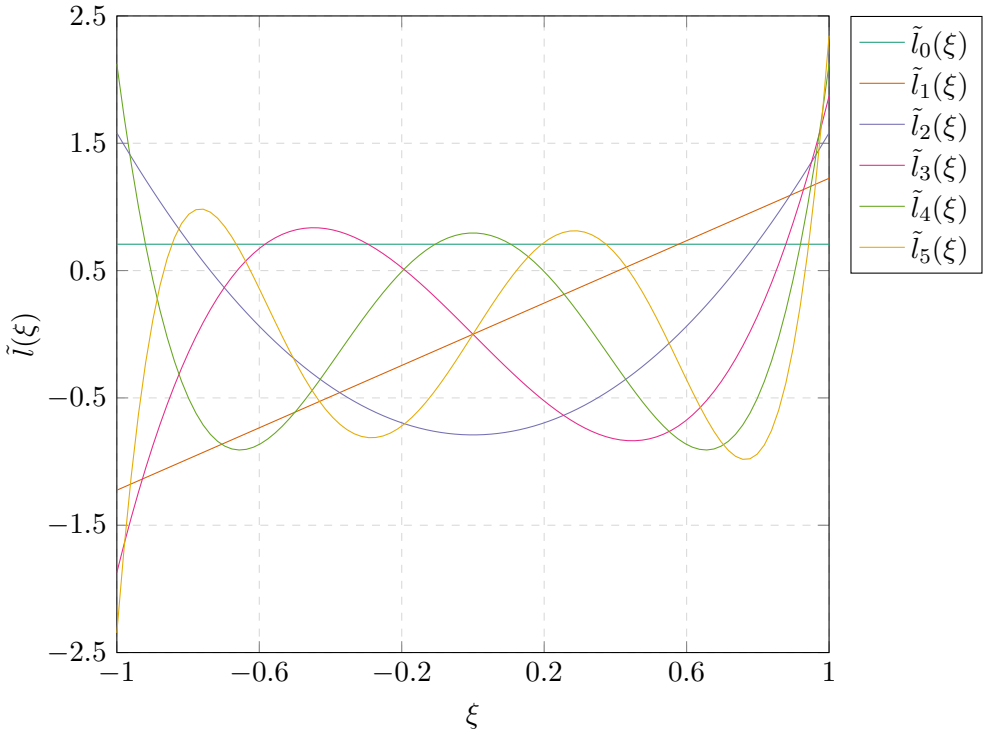

Figure 2.5. Legendre polynomials

2.3 Turbulence Modelling

The lack of a general solution to the Navier-Stokes equations means that engineers have to build mathematical tools in order to predict the evolution of turbulence. RANS, LES, and DNS are the most common approaches, and will be discussed below.

2.3.1 Direct Numerical Simulation

The DNS is the most computationally expensive approach. It consists of solving the Navier-Stokes equations, resolving all the scales of motion down to the dissipation scales. The relationship between the largest eddy and the smallest scale eddies is obtained from the Kolmogorov scale [**davidson2004turbulence**]:

$$\eta \sim lRe^{-\frac{3}{4}} = \left(\frac{\nu^3}{\epsilon}\right)^{\frac{1}{4}}, \tag{2.21}$$

where ν is the kinematic viscosity, η is the length scale of the smallest scale vortices, Re is the Reynolds Number, l is the scale of the largest eddy, and ϵ is the dissipation rate of the energy. Notice the exponential growth in the difference between the smallest and largest length scale with an increasing Reynolds Number. Indeed, the grid spacing, δx must be proportional to η to

capture the smallest eddies:

$$\Delta x \sim \eta \sim lRe^{-\frac{3}{4}}. \tag{2.22}$$

For a three-dimensional simulation, the required number of degrees of freedom is estimated at:

$$N_{DOF} = \left(\frac{L}{\Delta x}^3\right) \approx \left(\frac{L}{\eta}\right) Re^{\frac{9}{4}} \tag{2.23}$$

Where L is the edge length of a cubic domain of equal sides, and Δx is the characteristic grid-spacing. The resources required based on this Reynolds scaling is the reason DNS has not yet been widely adopted for industrial applications.

2.3.2 Large-Eddy Simulation

During the lifetime of an eddy, it will breakup through a progression of sizes. Due viscosity, the eddy is cascaded down into smaller sizes, until its kinetic energy is dissipated into heat [**davidson2004turbulence**]. The smallest eddies are often isotropic and are simpler to model for this reason. A visual explanation is given in Figure 2.6.

Implicit Large-Eddy Simulation

A SGS modelling approach called Implicit Large-Eddy Simulation (ILES) is adopted in this research. In ILES, we take advantage of the numerical error of FR to mimic the physical dissipation of the flow. In the case of FR, numerical dissipation concentrated at high wave numbers is effective at mimicing the dissipation of unresolved scales. Research has proven the validity of this approach in many validation studies when compared to DNS and experimental results [**VERMEIRE**].

LES resolves the large scale turbulence and uses a SGS to approximate the influence of small unresolved eddies on the large resolved eddies.

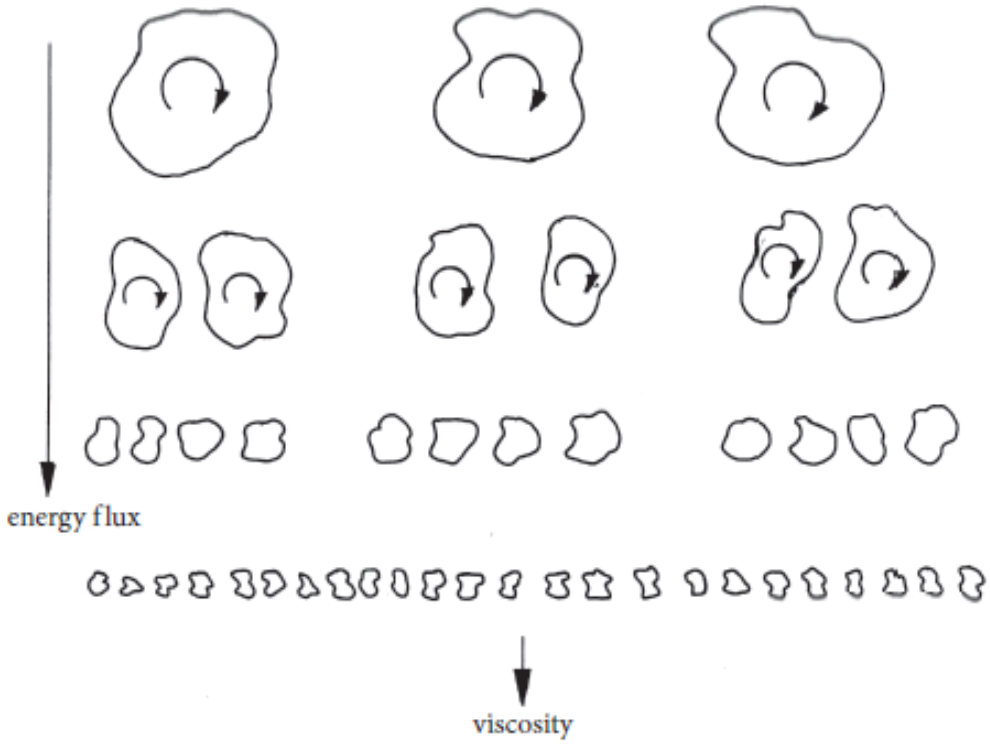

Figure 2.6. Schematic representation of energy cascade [**FrischBook**]

2.3.3 Reynolds-Averaged Navier-Stokes

The RANS approach has been widely adopted in general CFD for industrial applications due to its reasonable computational cost. By time-averaging the Navier-Stokes equations for incompressible fluids, we remove the necessity to resolve the higher-frequency range in the flow. This method was first introduced in a paper by Osborne Reynolds [**Osborne**]. Reynolds decomposition consists of splitting the velocity into two parts

$$u(x,t) = \overline{u(x)} + u(x,t)', \tag{2.24}$$

where $\bar{u}$ is the ensemble-averaged velocity and u' is the fluctuating component. The RANS equation for incompressible flow is is as follow [**Osborne**]

$$\rho\frac{\partial \bar{u}_i}{\partial t} + \rho\frac{\partial}{\partial x_j}(\bar{u}_i\bar{u}_j + \overline{u_i'u_j'}) = -\frac{\partial P}{\partial x_i} + \frac{\partial}{\partial x_j}(2\mu S_{ij}), \tag{2.25}$$

where S_{ij} is the mean strain-rate tensor

$$S_{ij} = \frac{1}{2}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right). \tag{2.26}$$

However, from the simplified time-averaged Navier-Stokes we have generated an unknown turbulent correlation, $\rho\overline{u_i' u_j'}$. for which there is no general solution. This requires the specification of an appropriate turbulence model for these so-called Reynolds stresses. Unlike LES, this turbulence mode must approximate the behaviour of all of the turbulent length scales, including both the small isotropic and large anisotropic ones. While several turbulence models have been proposed, they all have significant limitations, particularly in transitional and separated flows.

Reynolds-stress-transport Model

The most straightforward way is to model the Reynolds stresses to formulate a closure that determines them directly. Zhou (1945) has derived the time evolution of Reynolds stress [**Zhou1945**]:

$$\begin{aligned}\frac{\partial\overline{u_i' u_j'}}{\partial t} + \bar{u}_k\frac{\partial\overline{u_i' u_j'}}{\partial x_k} = &-\overline{u_i' u_k'}\frac{\partial\bar{u}_j}{\partial x_k} - \overline{u_j' u_k'}\frac{\partial\bar{u}_i}{\partial x_k} + \overline{\frac{p'}{\rho}\left(\frac{\partial u_i'}{\partial x_j} + \frac{\partial u_j'}{\partial x_i}\right)} - \\ &\frac{\partial}{\partial x_k}\left(\overline{u_i' u_j' u_k'} + \frac{\overline{p' u_i'}}{\rho}\delta_{jk} + \frac{\overline{p' u_j'}}{\rho}\delta_{ik} - \nu\frac{\partial\overline{u_i' u_j'}}{\partial x_k}\right) - \\ &2\nu\overline{\frac{\partial u_i'}{\partial x_k}\frac{\partial u_j'}{\partial x_k}}.\end{aligned} \tag{2.27}$$

We observe in the Equation above that by trying to find a more general equation for the Reynolds stresses we have obtained an expression with even more unknown, namely the triple correlations $\overline{u_i' u_j' u_k'}$. Another drawback is the low numerical stability associated with the absence of the numerically stabilizing second-order derivatives for the eddy-viscosity. Resulting in the necessity for measures to improve convergence [**Lien**].

Linear Eddy Viscosity Models

The RANS approach introduced a tensor known as the Reynolds Stresses. The symmetrical tensor has 6 unknowns. We may reduce the turbulence closure problem to 1 term using the Boussinesq approximation. The two equation model is used to solve the Boussinesq approximation. Where a turbulent viscosity is introduced that is responsible of the chaotic mixing and diffusion of the fluid. The parameter is determined from the local gradient

$$-\rho\overline{u_i' u_j'} = \underbrace{\mu_t\left(\frac{\partial\bar{u}_i}{\partial x_j} + \frac{\partial\bar{u}_j}{\partial x_i}\right)}_{deviatoric} - \frac{2}{3}\delta_{ij}\rho k. \tag{2.28}$$

The kronecker delta terms are corrections for realisability and the eddy viscosity terms account for the deviation to the average isotropic stresses [**schwartz**], i.e. the *deviatoric* parts of the stresses. To determine the viscosity, we know it must be a combination of a length scale and time scale, $\mu_t = lv$. Using the combination of each scale to represent the energetic level of the eddy of interest.

The popular $k-\epsilon$ model refers to the transport of the turbulent kinetic energy and the dissipation rate of turbulent kinetic energy as the respective velocity and length scale.

$$\mu_t = C_\mu \frac{k^2}{\epsilon} \tag{2.29}$$

The transport of k and ϵ are

$$\frac{\partial k}{\partial t} + \bar{u} \cdot \nabla k = -\nabla \cdot T' + P - \epsilon, \tag{2.30}$$

where T' is unknown and is approximated to $\left(\frac{\mu_t}{\sigma_k}\nabla k\right)$ [**Book:Pope**], P and ϵ are the production and dissipation terms in the kinetic energy equation. Discussed in later Section 3.2.

$$\frac{\partial k}{\partial t} + \bar{u} \cdot \nabla k = \nabla \cdot \left(\frac{\mu_t}{\sigma_k}\nabla k\right) + P - \epsilon. \tag{2.31}$$

To close the set of equations, the transport for dissipation rate ϵ must be defined. It is important to distinguish between the empirical and exact ϵ. The former is used rather than the exact equation because it is best viewed as the energy-flow rate in the cascade, determined by the large-scale motions [**Book:Pope**],

$$\frac{\partial \epsilon}{\partial t} + \bar{u} \cdot \nabla \epsilon = \nabla \cdot \left(\frac{\mu_t}{\sigma_\epsilon}\nabla \epsilon\right) + C_{\epsilon_1}\frac{P\epsilon}{k} - C_{\epsilon_2}\frac{\epsilon^2}{k}. \tag{2.32}$$

Where C_{ϵ_1} and C_{ϵ_2} are adjustable constants obtained through a comprehensive data fitting. Until now, the closure modeling is narrowed down to the calibration of constitutive relationships with canonical experimental data. Standard values for Launder and Sharma (1974) [**Book:Pope**] are

$$C_\mu = 0.09, \quad C_{\epsilon_1} = 1.44, \quad C_{\epsilon_2} = 1.92, \quad \sigma_k = 1, \quad \sigma_\epsilon = 1.3. \tag{2.33}$$

However, this approach has many shortcomings. For instance, the Boussinesq approximation [2.28] linearly relates the Reynolds stresses with the mean strain rate tensor. This will provide a very crude estimate when the flow has strong anisotropic turbulence [**davidson2004turbulence**]. Additionally, the Reynolds stresses can be visualized as a transient momentum mixing across the shear layer. Therefore, a more reliable model should reflect the eddy transition from the prior location rather than rely solely on local quantities. To reach better predictive accuracy, more advanced general eddy viscosity models have been proposed.

Explicit Algebraic Reynolds-stress Model

To improve from the linear eddy viscosity model, Pope (1975) developed the most general representation of the normalized anisotropic Reynolds stresses by deriving the relevant integrity bases [**pope_1975**].

$$b_{ij} = b_{ij}(\bar{s}, \bar{r}), \tag{2.34}$$

$$\bar{s}_{ij} = \frac{1}{2}\frac{k}{\epsilon}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right) = \frac{k}{\epsilon}\bar{S}_{ij}, \tag{2.35}$$

$$\bar{r}_{ij} = \frac{1}{2}\frac{k}{\epsilon}\left(\frac{\partial \bar{u}_i}{\partial x_j} - \frac{\partial \bar{u}_j}{\partial x_i}\right) = \frac{k}{\epsilon}\bar{R}_{ij}, \tag{2.36}$$

where $\bar{s}$ is the non-dimensional mean rate of strain and $\bar{r}$ is the non-dimensional rate of rotation. Based on the Cayley-Hamilton theory, where a matrix is the solution of its own characteristic equation. Pope and Gatski proved for three-dimensional flows the eddy viscosity can be expressed as a linear combination of 10 basis tensors and 5 irreducible invariants [**Gatski**]:

$$b = \sum_{n=1}^{10} g^{(n)}(\lambda_1, ..., \lambda_5)\boldsymbol{T}^{(n)}. \tag{2.37}$$

The coefficients $g^{(n)}$ in equation [2.37] are functions of a limited number of invariants. When an eddy viscosity model $g^{(n)} \neq 0$, the model is said to be non linear. This increases the accuracy of the solver with the added potential to capture better flow physics and streamline

curvature [**pope_1975**]. The five invariants are

$$\begin{aligned}
\lambda^{(1)} &= Tr(\bar{s}^2), \\
\lambda^{(2)} &= Tr(\bar{r}^2), \\
\lambda^{(3)} &= Tr(\bar{s}^3), \\
\lambda^{(4)} &= Tr(\bar{r}^2\bar{s}), \\
\lambda^{(5)} &= Tr(\bar{r}^2.\ \bar{s}^2)
\end{aligned} \tag{2.38}$$

The basis tensors are

$$\begin{aligned}
T^{(1)} &= \bar{s}, \\
T^{(2)} &= \bar{s}\bar{r} - \bar{r}\bar{s}, \\
T^{(3)} &= \bar{s}^2 - \frac{1}{3}\boldsymbol{I} \cdot Tr(\bar{s}^2), \\
T^{(4)} &= \bar{r}^2 - \frac{1}{3}\boldsymbol{I} \cdot Tr(\bar{r}^2), \\
T^{(5)} &= \bar{r}\bar{s}^2 - \bar{s}^2\bar{r}, \\
T^{(6)} &= \bar{r}^2\bar{s} + \bar{r}\bar{s}^2 - \frac{2}{3}\boldsymbol{I} \cdot Tr(\bar{s}\bar{r}^2), \\
T^{(7)} &= \bar{r}\bar{s}\bar{r}^2 - \bar{r}^2\bar{s}\bar{r}, \\
T^{(8)} &= \bar{s}\bar{r}\bar{s}^2 - \bar{s}^2\bar{r}\bar{s}, \\
T^{(9)} &= \bar{r}^2\bar{s}^2 - \bar{s}^2\bar{r}^2 - \frac{2}{3}\boldsymbol{I} \cdot Tr(\bar{s}^2\bar{r}^2), \\
T^{(10)} &= \bar{r}\bar{s}^2\bar{r}^2 - \bar{r}^2\bar{s}^2\bar{r},
\end{aligned} \tag{2.39}$$

where $\boldsymbol{I}$ is the three-dimensional identity tensor and $Tr(M)$ corresponds to the trace of the of tensor M.

In summary, the need for a modelling technique arises due to the lack of a general solution to the Navier-Stokes equations. RANS is the most popular approximation because of its fast computation time and lower computation cost. However, the Reynolds decomposition yields an unknown turbulent correlation known as the Reynolds Stresses. A widely used approach known as the Boussinesq approximation tackles the problem via Linear Eddy Viscosity. This way a turbulent viscosity is introduced, allowing for chaotic mixing and diffusion of the fluid.

Models like the $k - \epsilon$ are calibrated using a wide range of turbulent flows, but are inaccurate for transitional and turbulent separated flows. Instead of transporting turbulence quantities, we propose the use of deep neural network to estimate the amount of turbulent kinetic energy production and dissipation.

Chapter 3

Machine Learning Methodology

In this chapter, we will explain the multi-layer perceptron, an example implementation on the popular Iris dataset [**Iris**] using Tensorflow [**Tensorflow**], followed by the thesis framework and how the trained NN will fit within the CFD workflow.

3.1 Multi-Layer Perceptron (MLP)

The feedforward artificial neural network is the model of interest in this thesis. Unlike linear classification models, a neural network can optimize a decision surface for complex problems by adding multiple hidden layers. This is useful since most modern challenging problems are non-linear. To illustrate this, we project the $XOR(x_1, x_2)$ on a single neuron. We notice to the left of Figure 3.1, where no decision boundary can adequately separate the binary response. However, we can overcome this by applying a transformation on our input values as seen on the right of Figure 3.1.

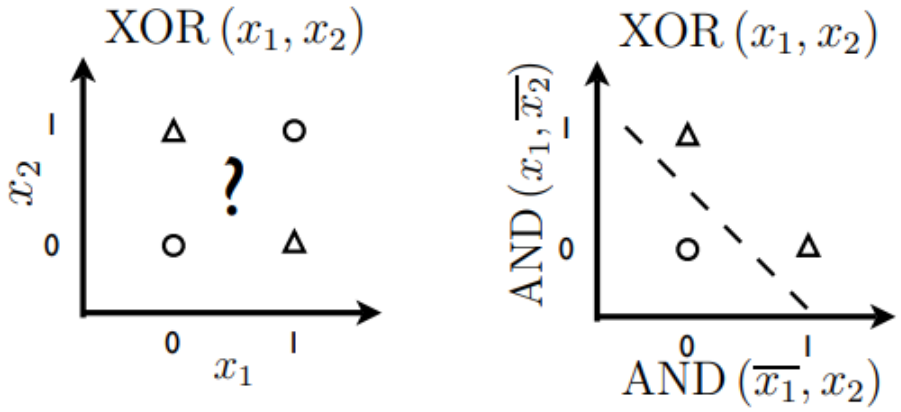

Figure 3.1. XOR gate modelisation [**hugo**]

Therefore, the inspiration behind deep learning or multi-layer perceptron is to find a new representation from the training data. The set of transformed input neurons are known as "hidden layers" because their values are not given in the data [**Goodfellow**].

$$\text{XOR}\,(x_1, x_2) = \begin{cases} 0 & \text{if } x_1 = 0 \text{ and } x_2 = 0 \\ 1 & \text{if } x_1 = 1 \text{ and } x_2 = 0 \\ 1 & \text{if } x_1 = 0 \text{ and } x_2 = 1 \\ 0 & \text{if } x_1 = 1 \text{ and } x_2 = 1 \end{cases} \tag{3.1}$$

Given an adequate amount of neural units on a single hidden layer, the artificial neural network has the potential to predict a variety of continuous non-linear function. This is the Universal approximation theorem, although there are some scepticism on whether it is practically feasible as it could require an incredibly large architecture [**Goodfellow**].

MLP Architecture

We begin the description of a Multi Layer Perceptron (MLP) with the focus on a single neuron. The first hidden layer unit in Figure 3.2 can be mathematically described as a mapping function

$$h^{\ell}(x)_i = g^{\ell}(a(x)_i) = g^{\ell}(b_i^{\ell} + \sum_{j=1}^{3} W_{i,j}^{\ell} x_j), \tag{3.2}$$

where ℓ is the hidden layer notation, b is the bias, W is the weight, g is the activation function and a is the pre-activation. Additionally, the index i is used to denote the neuron in the hidden layer. Furthermore, assuming a fully connected neural network, there's a weight for every pair of units between layers, hence $W_{i,j}$. The ℓth hidden layer notation are important due to the changes between layers. For instance, the activation function may change from one hidden layer to another.

The vector form is

$$h^{\ell}(x) = g^{\ell}(a(x)) = g^{\ell}(b^{\ell} + W^{\ell} x) \tag{3.3}$$

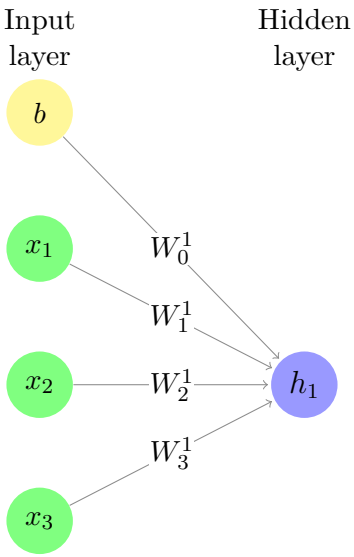

Figure 3.2. Single unit

As mentioned previously, the hidden layers purpose is to introduce transformations on the input data. Non-linearity in the model will help predict complex functions and this is achieved with the help of the activation function. A common activation function for binary classification is the sigmoid activation [**Liu2014ImageCF**].

$$\text{sigmoid}(x) = \sigma(x) = \frac{1}{a + e^{-x}}. \tag{3.4}$$

It is also useful to know the derivative of the sigmoid function

$$\text{sigmoid}'(x) = \sigma(x)(1 - \sigma(x)). \tag{3.5}$$

Finally, as shown on Figure 3.3, the output node is

$$f(x) = o(b^{(2)} + {W^{(2)}}^T h(x)), \tag{3.6}$$

where o is the output activation function. In general, a linear activation function is employed for the output. Softmax and the Rectified Linear Unit (ReLU) are used for classification and regression, respectively.

MLP Training

In supervised learning, we assume that we have a dataset that has both the input and its associated target. Therefore the performance of the model can be evaluated using a loss function

$$J = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}, y). \tag{3.7}$$

This allows us to cast training as our optimization problem. For instance, the gradient of the loss function with respect to the weight of the neural network is used to update the parameters after each iteration. This is known as the Stochastic Gradient Descent (SGD).

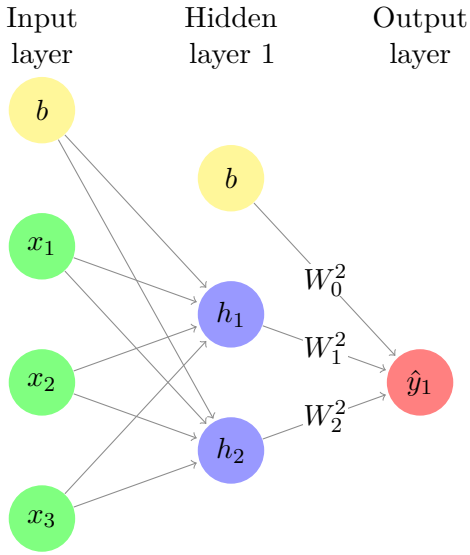

Figure 3.3. Schematic of MLP

$$\theta \equiv \{W_i^{(1)}, W_0^{(1)}, ..., W_i^{(\ell)}, W_0^{(\ell)}\}, \tag{3.8}$$

$$\triangle = -\nabla_\theta \mathcal{L}(f(x;\theta), y). \tag{3.9}$$

where θ is the set of all weights in the neural network and $\triangle$ is the loss derivative with respect to the weights. The result of the loss function gradient with respect to the parameter θ is expressed in the above equation. Another term for this is Back Propagation. Where $f(x;\theta)$ is the output of the neural network. Given this, updated weights can be found via

$$\theta \leftarrow \theta + \alpha \triangle, \tag{3.10}$$

where α is the learning rate and it controls by how much the parameter should change during each iteration. The learning rate is a hyper-parameter that helps the loss function to converge into a local minima. How after each iteration as the SGD approaches the minima, it can over shoot if the learning rate is too large. Alternatively, it can slow down the training process if the learning rate is too small. There are common values for learning rates but there is no values that fits all. A paper by Leslie N. Smith proposed a systematic approach, where one would gradually increase the learning rate for each mini-batch until the loss starts to explode [**CyclicalSmith**]. Once the algorithm has trained over all given instances, we say the training reached one epoch. Ideally, by the end of the training, the model will have optimal values of weight and can accurately generalize on unseen data.

As a practical example, if we would take the gradient of the loss function with respect to $W^{(2)}$ as seen from figure 3.3. The chain rule will be applied

$$\frac{\partial \mathcal{L}}{\partial W^{(2)}} = \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial a} \frac{\partial a}{\partial W^{(2)}}, \tag{3.11}$$

$$\frac{\partial y}{\partial a} = \frac{\partial \sigma(a)}{\partial a} = y(1-y), \tag{3.12}$$

$$\frac{\partial a}{\partial W^{(2)}} = \frac{\partial \left(W^{(2)} \sigma^{(L-1)} + W_0^{(2)} \right)}{\partial W^{(2)}} = \sigma^{(L-1)}, \tag{3.13}$$

where y is the activated output, and a is the pre-activation. Therefore, our first derivative will depend on the type of loss function. We also notice the second derivative of the activated output with respect to the pre-activation is only the derivative of the sigmoid, Equation 3.5. Furthermore, the third and last derivative is nothing more but the activated output from the previous layer (hidden layer).

3.1.1 Iris Dataset

In this section, we will showcase the conventional approach to train a neural network. The demonstration will be delivered using one of the most common studied datasets, the Iris

dataset[**Iris**, **UCI**].

Data Preparation

The dataset contains 50 samples from three species of the Iris plants (Iris setosa, Iris virginica and Iris versicolor). Four features are measured for each instance, namely the length and the width of the sepals and petals. Prior to the ML model construction, the data explanatory analysis is a crucial step. Not only do we gain a more comprehensive understanding of the dataset, we can also apply transformation to improve the quality of our data.

	sepal length	**sepal width**	**petal length**	**petal width**
count	150.00	150.00	150.00	150.00
mean	5.84	3.06	3.76	1.193
std	0.83	0.44	1.77	0.76
min	4.30	2.00	1.00	0.10
25%	5.10	2.80	1.60	0.30
50%	5.80	3.00	4.35	1.30
75%	6.40	3.30	5.10	1.80
max	7.90	4.40	6.90	2.50

Table 3.1. Iris dataset statistics

There is no missing input and the target distribution is also well balanced. Therefore, we can avoid feature engineering tasks. Out of the total 150 instances, 10% will be split into the test dataset. On Figure 3.4, we observe the correlation amongst the different features. We conclude a high positive correlation for the petal features with the exception to the sepal width.

Training Model

Three models have been trained, each step will showcase an improvement over the previous model. In Figure 3.5, the model is composed of 4 layers with 128 units and 5 layers of 64 units. All units are initialized with He Uniform [**HeUniform**], the bias are initially one, and the activation function is ReLU[**relu2018**]. The output layer is a softmax function with 3 units. To accelerate training, it is good practice to use mini-batches, which is set to 40. Additionally, the Adam optimization algorithm [**adam2014**] with an initial learning rate of 10^{-4} was used. A 15% validation split is set aside during each epoch to monitor the training status. We notice a clear indication of over-fitting on the in loss in Figure 3.5. This happens when the model learns the noise in a limited training data set, which it is unable to generalize on unseen data.

To reduce undesirable overfitting, we add regularization techniques to the model. The weight

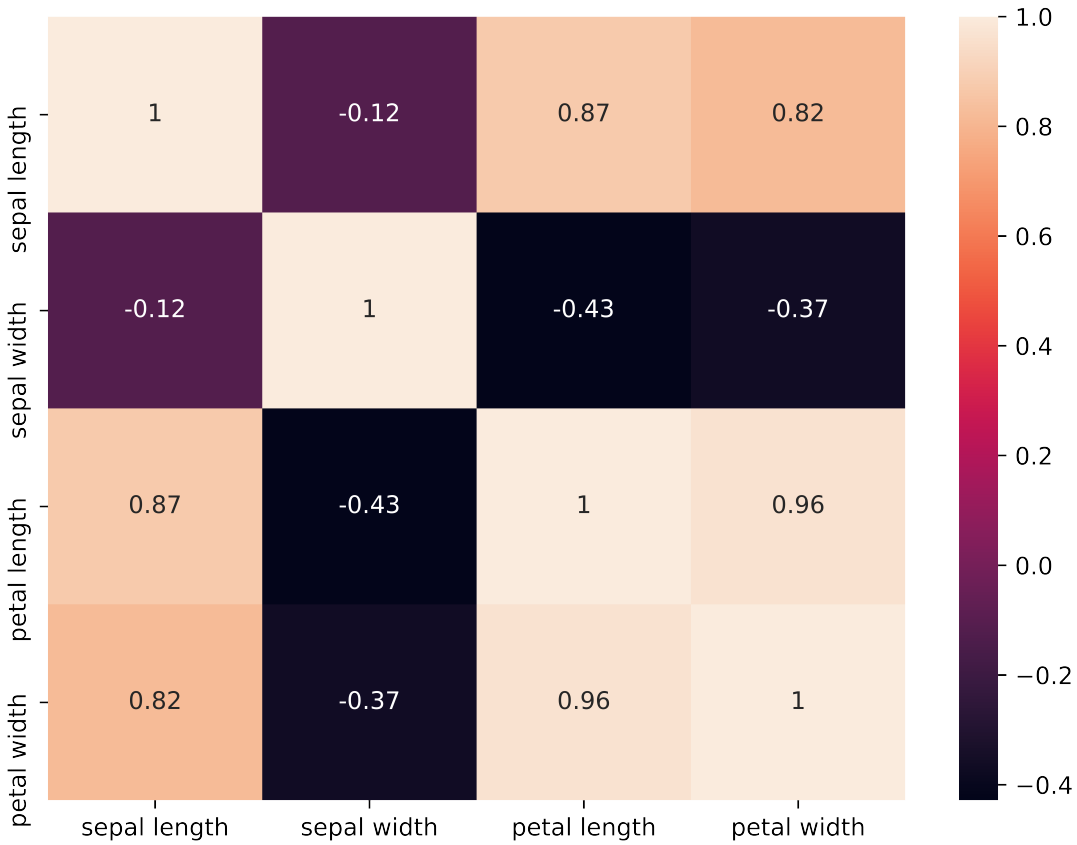

Figure 3.4. Iris heatmap

decay and dropout [**dropout2014**] are implemented in the second model in Figure 3.6. Here, the L_2 method adds a penalty to the cost function, J. This has the effect of limiting the changes in weight matrices from the hidden layers. In turn, this process simplifies the model to achieve better results. The modified loss function is

$$J = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}, y) + \frac{\lambda}{2m}\sum_{\ell=1}^{L}\|W^{[\ell]}\|^2, \tag{3.14}$$

where λ is the regularization parameter. Also, m stands for the number of training instances, and L is the number of hidden layers.

The validation loss in figure 3.6 does not diverge from the training loss anymore. This is a clear indication that the regularization technique has successfully reduce the overfitting in the network.

To further improve the model, we can introduce early-stopping. This feature informs the

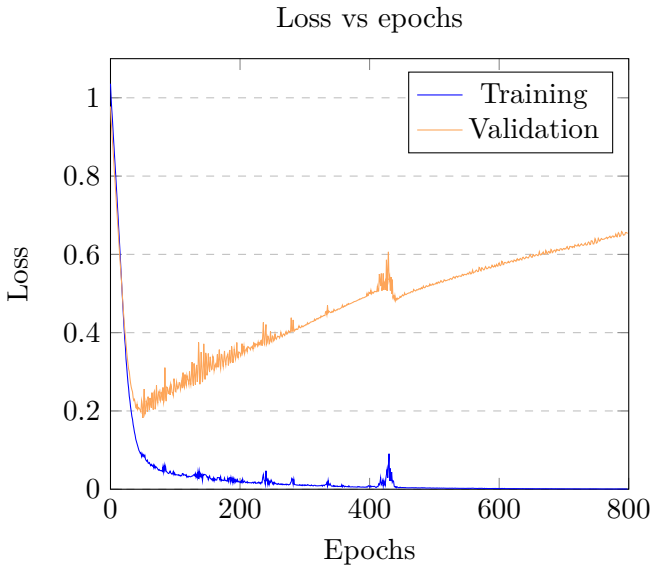

Figure 3.5. Iris dataset performance

neural network to stop the training once the validation loss does not improve after 30 epochs. The benefit is to have an accurate model trained in a shorter amount of time. As shown on Figure 3.7, training terminated once the validation loss stopped improving. The same techniques will be applied in the numerical experiments in Chapter 4.

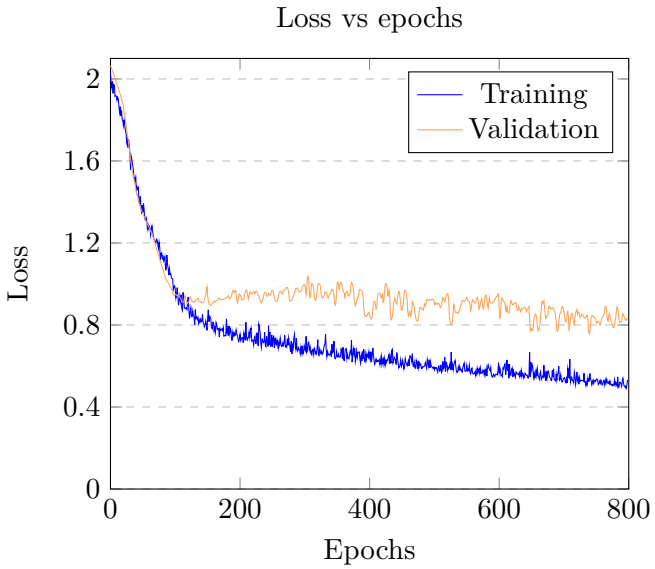

Figure 3.6. Iris dataset performance with regularization

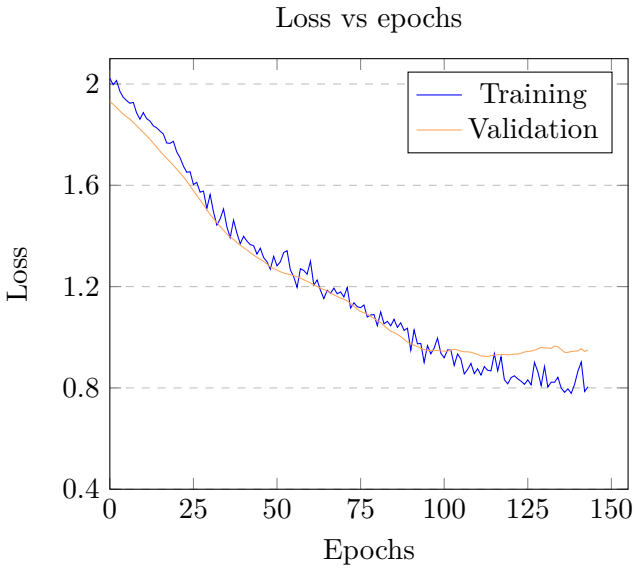

Figure 3.7. Iris dataset performance with regularization and early-stopping

3.2 Framework

Using local quantities, we leverage machine learning capabilities to improve the eddy viscosity prediction. Currently, most popular RANS solvers marches the turbulent kinetic energy (TKE) field in time and space to calculate the Reynolds stresses, such as the $k-\omega$ and $k-\epsilon$ model. The TKE and the Reynolds stresses are related by the Boussinesq approximation

$$\tau_{ij}^{R} = -\rho\overline{u_i' u_j'} = \mu_t \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right) - \frac{2}{3}\delta_{ij}\rho k. \tag{3.15}$$

The Reynolds stresses, τ_{ij}^{R}, and the mean strain rate, $\overline{S}_{ij}$, are related by the scalar eddy viscosity, μ_t. The latter is the term of interest, its accuracy depends on 3 elements. Mainly, the constitutive relations, the predicted velocity and length scale. However, the closure models are calibrated with canonical cases. Instead of using calibrated model constants, we can train the neural network to locally approximate the exact TKE production and dissipation values. In essence, the TKE partial differential equation is replaced by the machine learning model. Notice the term D in Equation 3.16 is the second transported variable in the $k-\epsilon$ model.

The full form of the TKE equation is

$$\frac{\partial \bar{\rho} k}{\partial t} + \frac{\partial \bar{\rho} k \tilde{u}_i}{\partial x_i} = A + B + C + D, \tag{3.16}$$

where

$$A = -\overline{\rho u_i'' u_j''} \frac{\partial \tilde{u}_i}{\partial \tilde{u}_j}, \tag{3.17}$$

$$B = \frac{\partial}{\partial x_j}\{ -\frac{1}{2}\overline{\rho u_i'' u_i'' u_j''} + \overline{u_i'' \tau_{ij}} - \overline{p u_j''}\}, \tag{3.18}$$

$$C = \overline{p \frac{\partial u_i''}{\partial x_i}}, \tag{3.19}$$

$$D = \overline{\tau_{ij} \frac{\partial u_i''}{\partial x_j}}. \tag{3.20}$$

Term	Description
A	Production
B	Diffusion
C	Pressure
D	Dissipation

Table 3.2. Terms in the Turbulence kinetic energy equation

The present work seeks to complement the eddy-viscosity approach with a trained neural network for reliable estimation of the TKE production and dissipation values. High-fidelity data from a DNS simulation feeds into the training pipeline. As a reminder, the model works in the RANS environment. Thus, only the same available input data in the RANS environment are used to train the neural network. The turbulent channel case is a 1 dimensional flow and the available features are

$$y/d,\ \overline{\rho},\ \overline{\rho u_i},\ \overline{\rho E},\ \overline{\frac{\partial \rho}{\partial x_i}},\ \overline{\frac{\partial \rho u_i}{\partial x_i}}. \tag{3.21}$$

3.2.1 Dataset Target Variables

This section will address how the TKE production and dissipation are extracted from the DNS. The approach relies on the Favre-averaging for fluid compressibility effects.

Favre Averaging

To advance physical understanding and accuracy, we must rely on high-fidelity data. Favre-averaging does so by accurately capturing fluid density changes in time averaged fields. It is important to mention the focus will be on fully transitioned fluid flow without chemical reactions or real gas effects. Similar to the Reynolds decomposition, Favre-averaging also has a mean and fluctuating component. Some useful rules of Favre averaging are

$$\tilde{f} = \frac{\overline{\rho f}}{\bar{\rho}}, \tag{3.22}$$

$$f = \tilde{f} + f'', \tag{3.23}$$

$$\tilde{\bar{f}} = \tilde{f}, \tag{3.24}$$

$$\overline{\rho f''} = 0. \tag{3.25}$$

Production Term

The production is generated through the interaction between the shear stress and shear strain within the flow. Using the stated rules, we can deconstruct the production term into variables we can be extracted from the DNS.

$$A = \underbrace{-\overline{\rho u_i'' u_j''}}_{\text{part 1}} \underbrace{\frac{\partial \tilde{u}_i}{\partial x_i}}_{\text{part 2}}, \tag{3.26}$$

$$\underbrace{-\overline{\rho u_i'' u_j''}}_{\text{part 1}} = -\overline{\rho u_i u_j} + 2\overline{\rho u_i}\tilde{u}_j - \tilde{\rho}\tilde{u}_i\tilde{u}_j = -\overline{\rho u_i u_j} + 2\bar{u}_i\overline{\rho u_j} - \bar{\rho}\frac{\overline{\rho u_i}}{\bar{\rho}}\frac{\overline{\rho u_j}}{\bar{\rho}}, \tag{3.27}$$

$$\underbrace{\frac{\partial \tilde{u}_i}{\partial x_i}}_{\text{part 2}} = \frac{\partial(\frac{\overline{\rho u_i}}{\bar{\rho}})}{\partial x_j}. \tag{3.28}$$

Applying the quotient rule on the numerator in Equation 3.28

$$\frac{\partial \tilde{u}_i}{\partial x_j} = \frac{\bar{\rho}\frac{\partial \overline{\rho u_i}}{\partial x_j} - \overline{\rho u_i}\frac{\partial \bar{\rho}}{\partial x_i}}{\bar{\rho}^2}. \tag{3.29}$$

Applying the product rule on the first numerator in Equation 3.29.

$$\frac{\partial \tilde{u}_i}{\partial x_j} = \frac{\bar{\rho}\left[\overline{\frac{\partial \rho}{\partial x_j}u_i} + \overline{\rho\frac{\partial u_i}{\partial x_j}}\right] - \overline{\rho u_i}\frac{\partial \bar{\rho}}{\partial x_i}}{\bar{\rho}^2}. \tag{3.30}$$

Finally, the production term is the result of Equation 3.27 and Equation 3.30,

$$A = \left(-\overline{\rho u_i u_j} + 2\bar{u}_i\overline{\rho u_j} - \bar{\rho}\frac{\overline{\rho u_i}}{\bar{\rho}}\frac{\overline{\rho u_j}}{\bar{\rho}}\right) \times \frac{\bar{\rho}\left[\overline{\frac{\partial \rho}{\partial x_j}u_i} + \overline{\rho\frac{\partial u_i}{\partial x_j}}\right] - \overline{\rho u_i}\frac{\partial \bar{\rho}}{\partial x_i}}{\bar{\rho}^2}. \tag{3.31}$$

Dissipation Term

Following the same systematic approach for the Production term. We define the laminar stress tensor

$$\tau_{ij} = \lambda \frac{\partial u_k}{\partial x_k} \delta_{ij} + \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right), \tag{3.32}$$

where $\lambda = -\frac{2}{3}\tau$ and τ is the molecular viscosity which is a function of static temperature. Using Einstein notation

$$D = \overline{\tau_{ij} \frac{\partial u_i''}{\partial x_j}} = \overline{\left(\lambda \frac{\partial u_k}{\partial x_k} \delta_{ij} + \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \right) \frac{\partial u_i''}{\partial x_j}}. \tag{3.33}$$

As a reminder

$$\frac{\partial u_i''}{\partial x_j} = \frac{\partial \bar{u}_i}{\partial x_j} - \frac{\partial \tilde{u}_i}{\partial x_j}. \tag{3.34}$$

The final dissipation equation is

$$D = \overline{-\frac{2}{3}\mu \frac{\partial u_k}{\partial x_k} \delta_{ij} \frac{\partial u_i}{\partial x_j}} + \overline{\frac{2}{3}\mu \frac{\partial u_k}{\partial x_k} \delta_{ij}} \frac{\partial \tilde{u}_i}{\partial x_j} + \mu \overline{\frac{\partial u_i}{\partial x_j} \frac{\partial u_i}{\partial x_j}} - \mu \frac{\partial \bar{u}_i}{\partial x_j} \frac{\partial \tilde{u}_i}{\partial x_j} + \mu \overline{\frac{\partial u_i}{\partial x_j} \frac{\partial u_j}{\partial x_i}} - \mu \frac{\partial \bar{u}_i}{\partial x_j} \frac{\partial \tilde{u}_j}{\partial x_i}. \tag{3.35}$$

Nondimensionalizing Target Variables

To validate the work, we have to nondimensionalize the data as per K.M.M. [**KMM**].
Starting with equation 3.26,

$$A = -\overline{\rho u_i'' u_j''} \frac{\partial \tilde{u}_i}{\partial x_i} \left[\frac{kg}{m \cdot s^3} \right]. \tag{3.36}$$

Similarly for Dissipation with equation 3.2.1,

$$D = \overline{\tau_{ij} \frac{\partial u_i''}{\partial x_j}} \left[\frac{kg}{m \cdot s^3} \right]. \tag{3.37}$$

From K.M.M. documentation [**KMM**], they used the kinematic viscosity over the frictional velocity, ν/u_τ. The Nondimentionalizing Factor (NF) has been found to be the following,

$$NF = \frac{\mu}{\rho u_\tau^4} = \frac{\nu}{\rho^2 u_\tau^4} \left[\frac{m \cdot s^3}{kg} \right]. \tag{3.38}$$

Chapter 4

Numerical Experiments

4.1 Turbulent Channel

4.1.1 Case Description

One of the first DNS ever performed was a turbulent channel case [**KMM**]. This fully developed flow is quasi-one dimensional and is driven by a specified pressure gradient, dp/dx, in the streamwise direction. The channel is 2δ tall and is enclosed between two infinite plate in the $x-z$ plane. The dataset is generated using the PyFR DNS are performed at five friction Reynolds number $\mathrm{Re}_\tau = [180, 285, 395, 450, 590]$.

The case with friction Reynolds number $Re_\tau = u_\tau \delta/\mu = 395$, has a bulk Reynolds number Re_b=13773 which was found using *Deans' law* [**Book:Pope**]. The fluid density and kinetic viscosity are denoted by ρ and μ respectively. The flow fields start laminar and will transition to turbulence after sufficient time. However, the transition process to turbulence in an uniform flow can be very slow without upstream agitation. Therefore, we can accelerate the transition by adding perturbation in the initial condition.

$$u = -11.2 \cdot y \cdot (y-2), \tag{4.1}$$

$$v = 0.5 \cdot sin(5.5 \cdot \pi y) \cdot 0.5 \cdot sin(5.5 \cdot \pi x), \tag{4.2}$$

$$w = 0.5 \cdot sin(5.5 \cdot \pi z) \cdot 0.5 \cdot sin(5.5 \cdot \pi y), \tag{4.3}$$

where the bulk velocity v_b is given by

$$v_b = \int_0^{\delta} \bar{v} d\left(\frac{y}{\delta}\right) \approx 6.8. \tag{4.4}$$

Near the center of the channel, the values of TKE production and dissipation reach very small values and possibly spurious due to the PyFR solver. From observation, the ratio of production and dissipation is close to null. A corrolary is that the cross-correlations vanishes in strain-free flow region which is the center part of the turbulent channel. Thus, only the region prior to a clipping factor are kept, $y < 0.8\delta$ and $y > 1.2\delta$. Both correspond to the bottom and top region respectively. To ensure the simulation has reached steady state, the friction reynolds has been tracked over time. As shown on Figure 4.1, the channel has reached the desired $\mathrm{Re}_\tau = 395$ after 70 convective times. A convective time is defined as the time taken for the flow to travel across the channel.

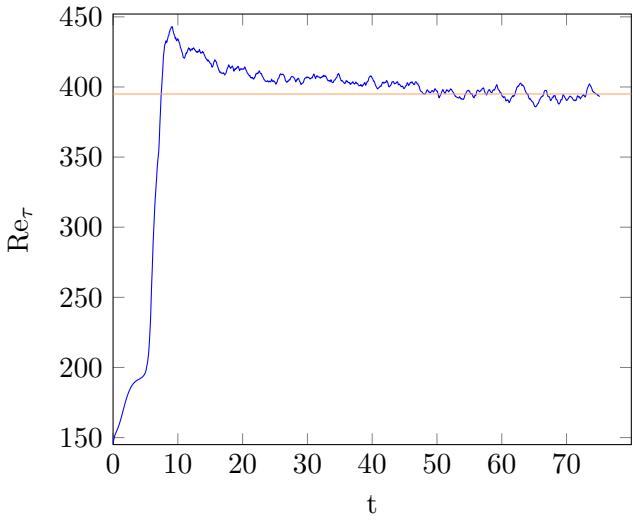

Figure 4.1. Friction Reynolds number development over time

Figure 4.2, 4.3, and 4.4 show the isosurfaces of Q-criterion for Re = 180, 395, and 590. They are colored by velocity magnitude. As expected, all simulations manifest vortical structures along the bottom plane. We observe difference in the results, the size of vortices are finer for larger Reynolds values.

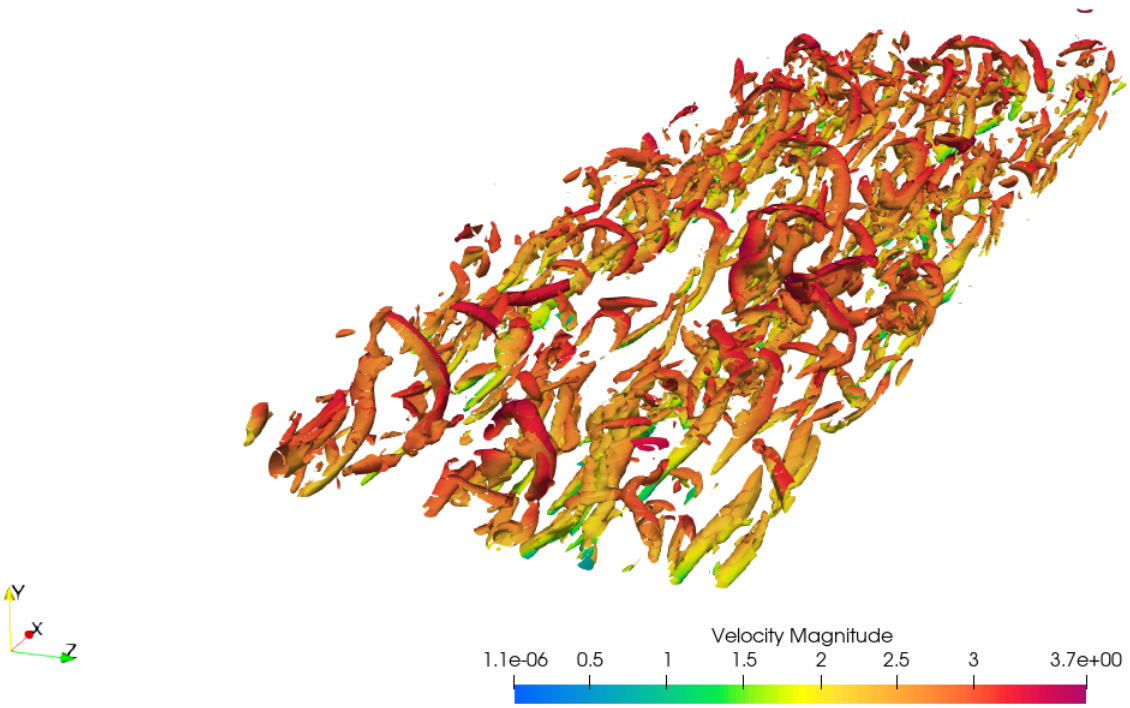

Figure 4.2. Q-criterion for Re = 180

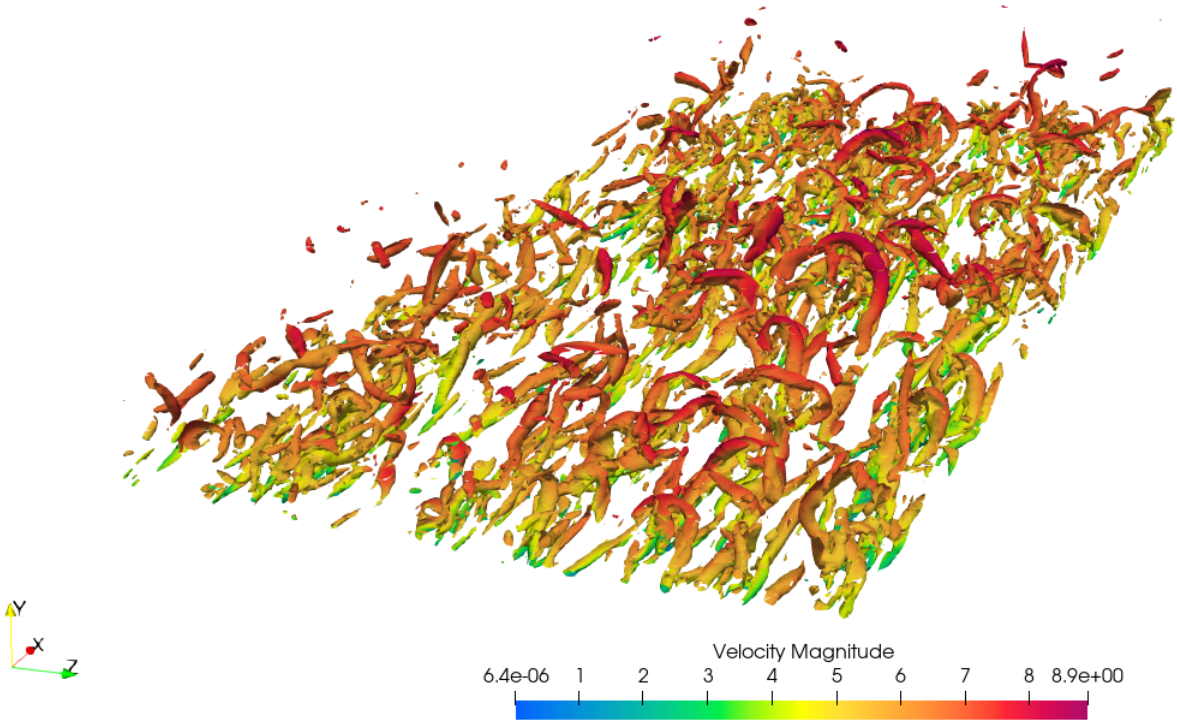

Figure 4.3. Q-criterion for Re = 395

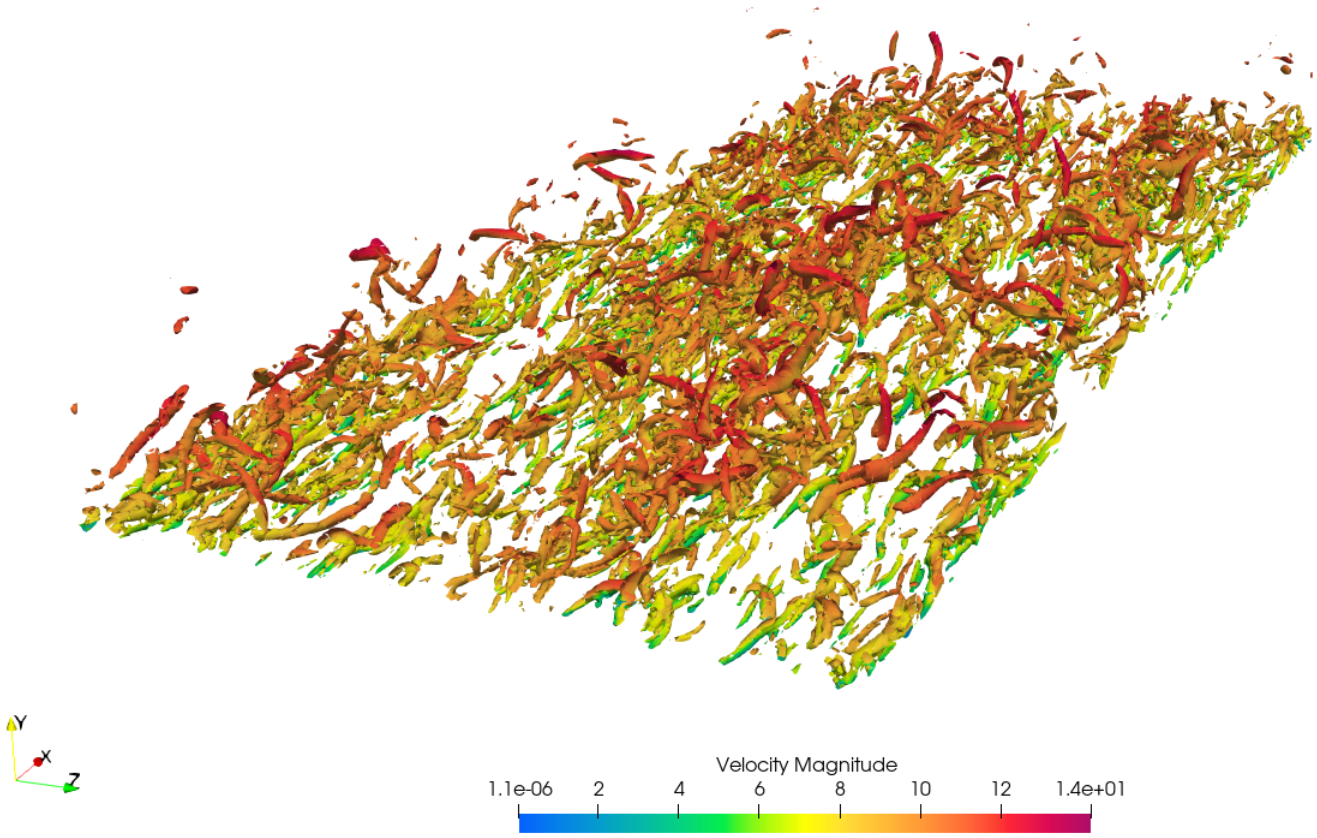

Figure 4.4. Q-criterion for Re = 590

We will now consider the accuracy of the simulated turbulent channel. We observe excellent agreement for normalized streamwise velocity with the results of K.M.M [**KMM**] for all Reynolds number in Figure 4.10, 4.16, and 4.22. The root-mean-square velocity fluctuations shows good agreement in Figure 4.9, and 4.15. However, in Figure 4.21 the DNS has larger numerical errors than the previous two cases. This slight difference is also reflected in the underprediction for the Reynolds stress, $\overline{u'v'}$, in Figure 4.20. The production and dissipation are accurately captured and are at the center core of this thesis. They are shown on Figure 4.5, 4.6, 4.11, 4.12, 4.17, and 4.18.

In summary, the flow properties and turbulence statistics of our DNS have been validated using the data found in the litterature [**KMM**]. This will allow to safely use our model to generate new data points at other Reynolds number.

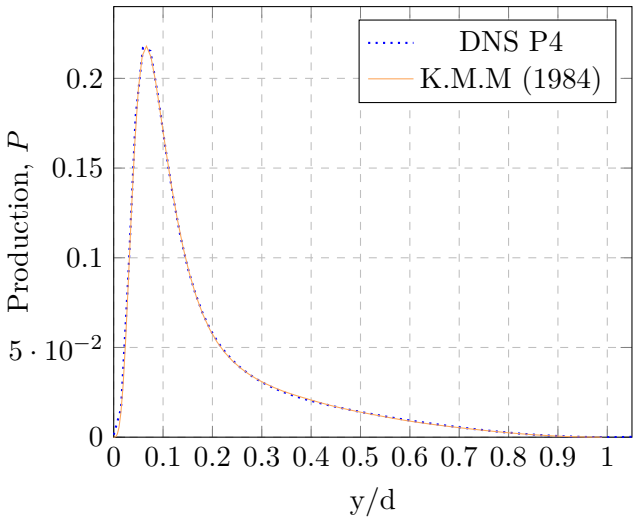

Figure 4.5. Production at $\mathrm{Re}_\tau = 180$

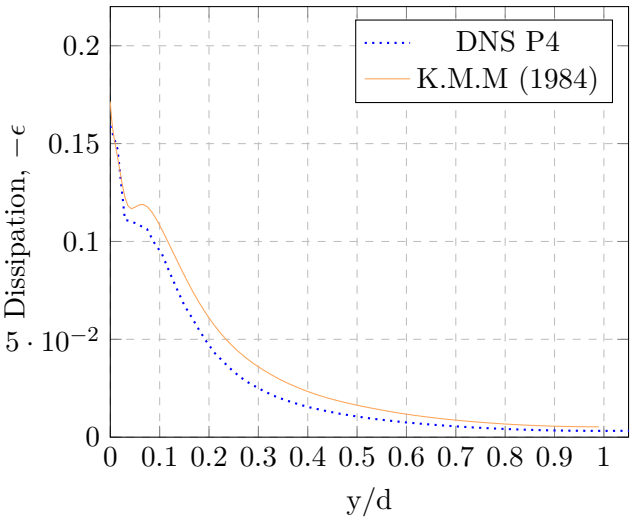

Figure 4.6. Dissipation at $\mathrm{Re}_\tau = 180$

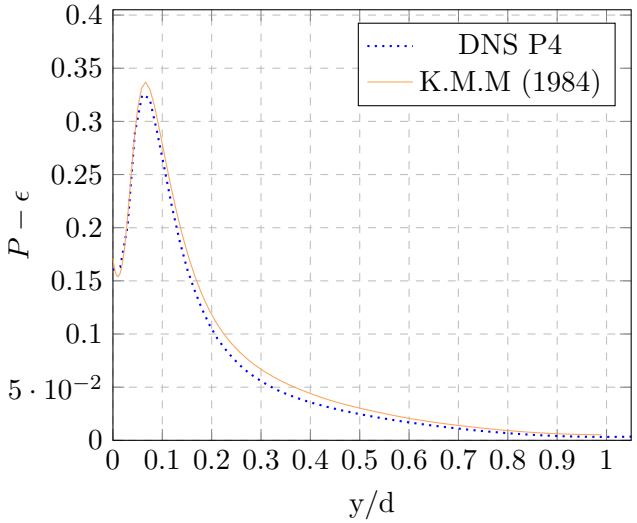

Figure 4.7. Production - Dissipation $\mathrm{Re}_\tau = 180$

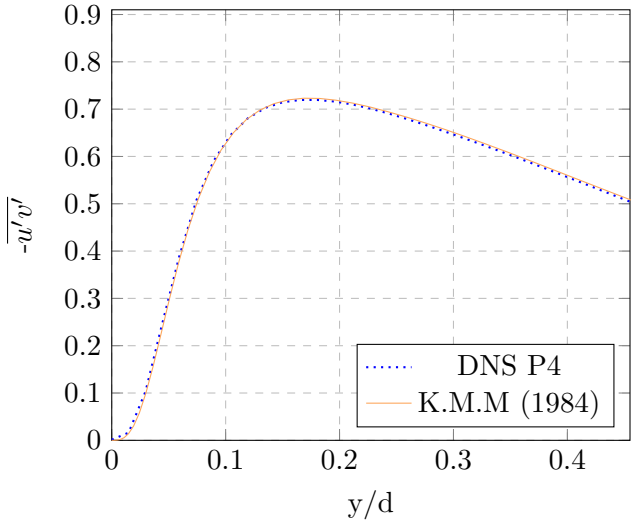

Figure 4.8. Reynolds Stress $\mathrm{Re}_\tau = 180$

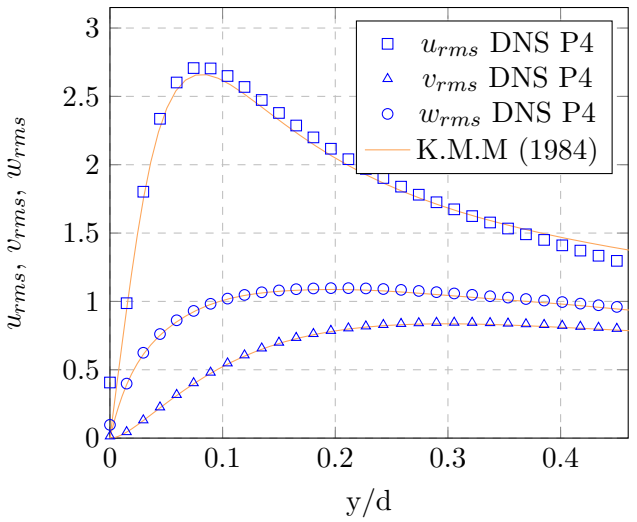

Figure 4.9. Velocity Fluctuations $\mathrm{Re}_\tau = 180$

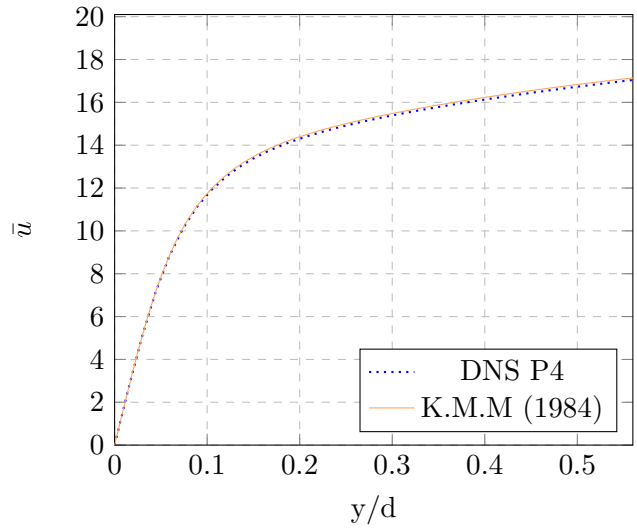

Figure 4.10. Velocity Profile $\mathrm{Re}_\tau = 180$

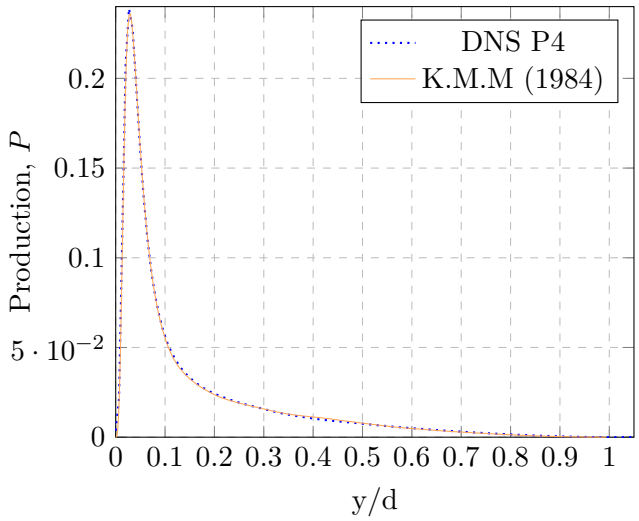

Figure 4.11. Production at $\mathrm{Re}_\tau = 395$

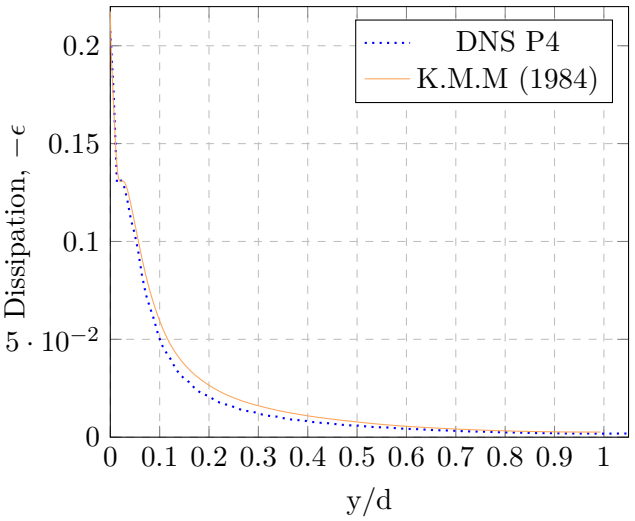

Figure 4.12. Dissipation at $\mathrm{Re}_\tau = 395$

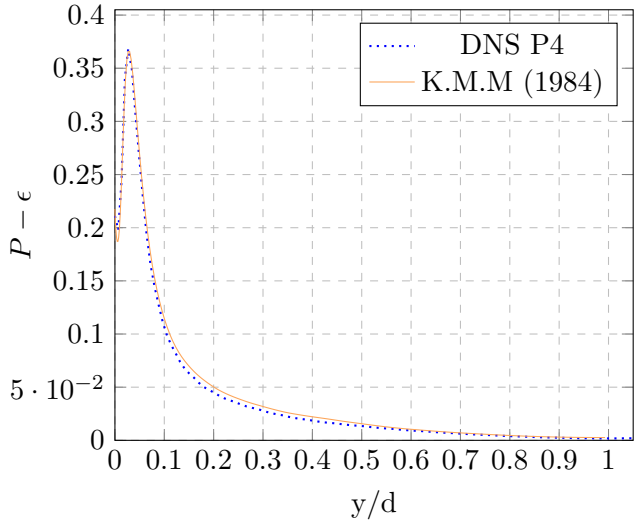

Figure 4.13. Production - Dissipation $\mathrm{Re}_\tau = 395$

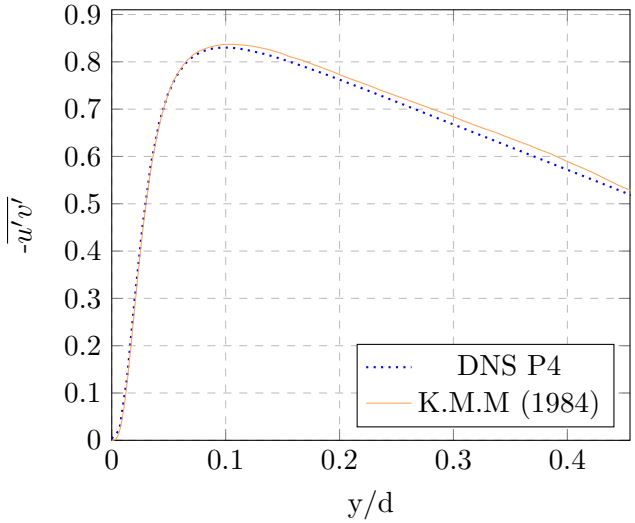

Figure 4.14. Reynolds Stress $\mathrm{Re}_\tau = 395$

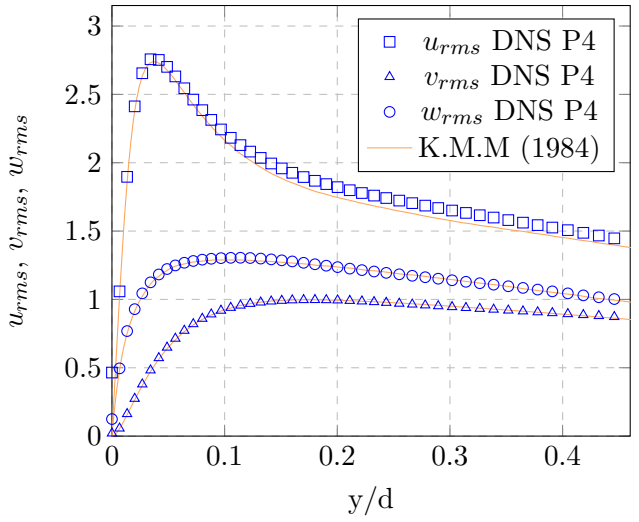

Figure 4.15. Velocity Fluctuations $\mathrm{Re}_\tau = 395$

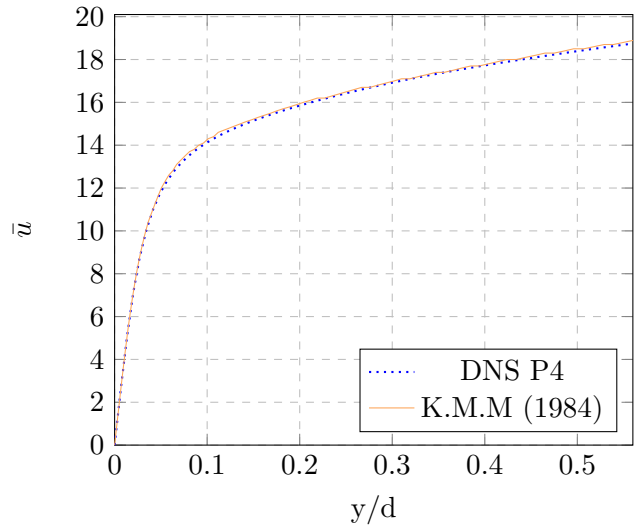

Figure 4.16. Velocity Profile $\mathrm{Re}_\tau = 395$

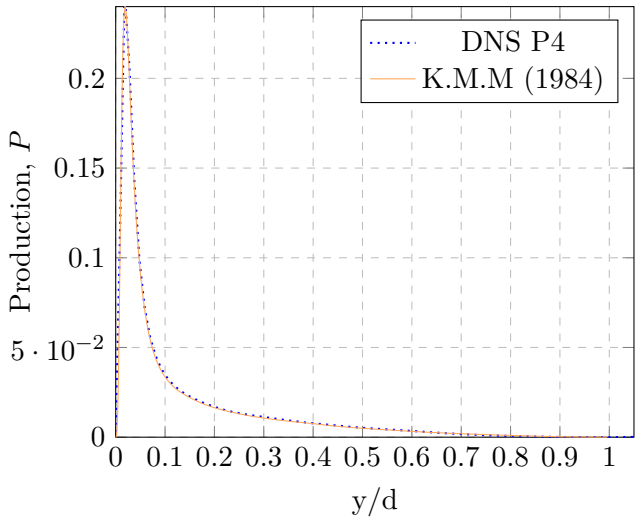

Figure 4.17. Production at $\mathrm{Re}_\tau = 590$

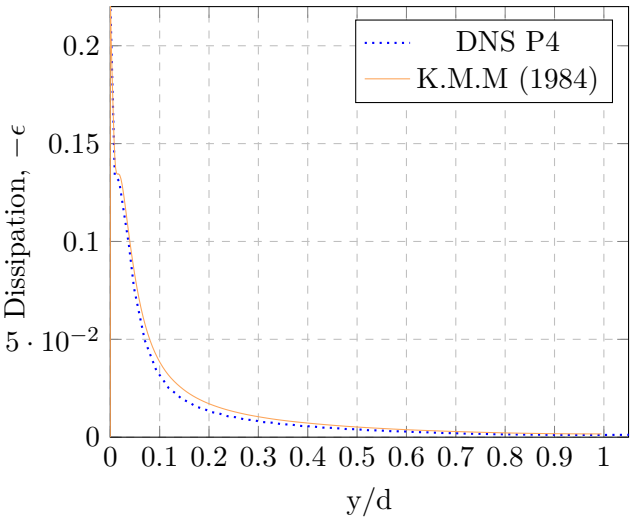

Figure 4.18. Dissipation at $\mathrm{Re}_\tau = 590$

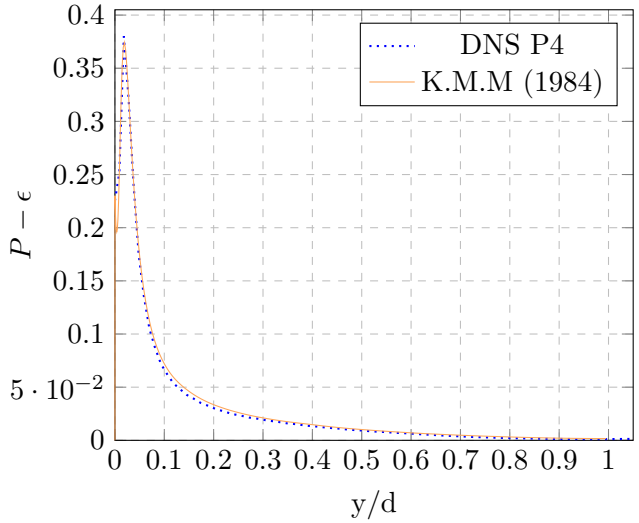

Figure 4.19. Production - Dissipation $\mathrm{Re}_\tau = 590$

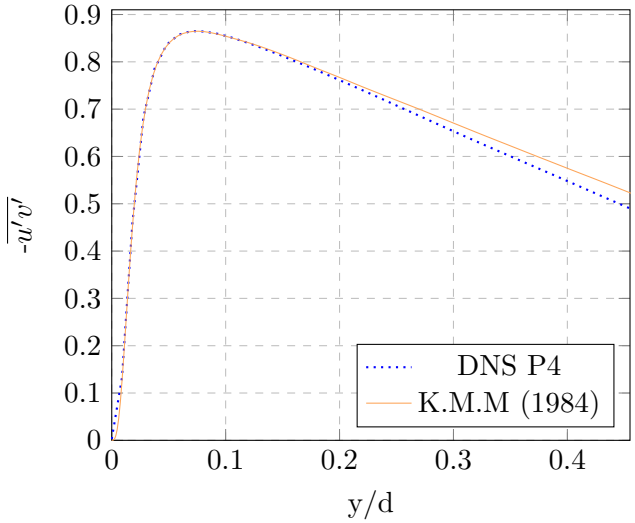

Figure 4.20. Reynolds Stress $\mathrm{Re}_\tau = 590$

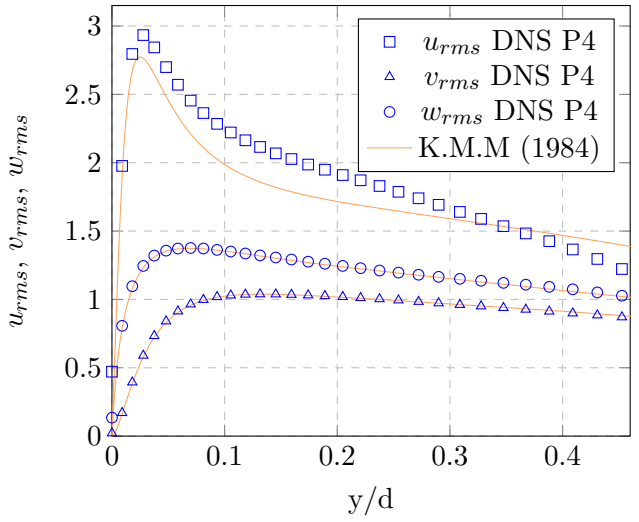

Figure 4.21. Velocity Fluctuations $\mathrm{Re}_\tau = 590$

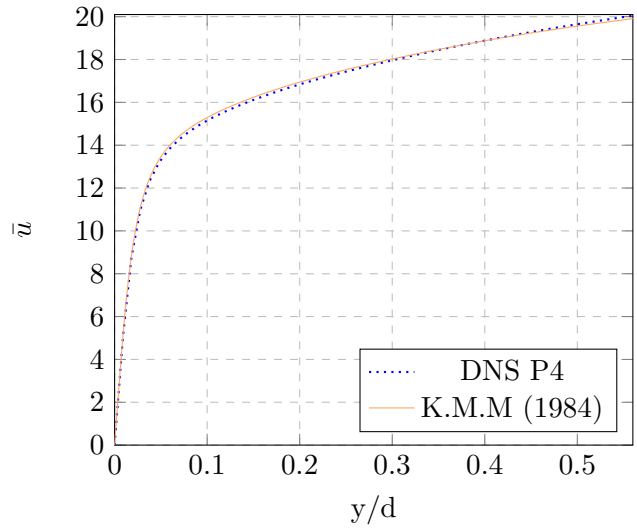

Figure 4.22. Velocity Profile $\mathrm{Re}_\tau = 590$

4.1.2 Machine Learning Framework

The dataset consists of 56 features, 1 continuous output, and 32 325 instances. All features are complete with no missing values. Notice the high number of features. To filter out the unnecessary features, we must remember the end goal of the machine learning approach; to approximate the TKE production and dissipation. This means we are limited to the same inputs as exist in the TKE equation. In other words, the dataset (production and dissipation) are extracted from the DNS of the Turbulent Channel. As such, the Reynolds stress is known but not in the RANS in framework. Once the features are determined, data transformation by normalization is applied since it has been shown to speed up convergence via gradient descent and implicitly balances the contribution of all features.

For each case, we randomly split the training data into 80% training and 20% validation data. The loss function used is the MSE between the predicted production and the DNS value. The Adam optimization [**adam2014**] and ReLU activation [**relu2018**] are used with an initial learning rate of 10^{-6} and a batch size of 10. The weights of the networks were initialized with He uniform [**HeUniform**] and biases were initialized as zeros. To prevent over-fitting our model, a regularization callback is imposed on the loss function and batch normalization is implemented to accelerate the training.

The proposed cases to evaluate the suitability of a neural network are shown in Table 4.1. From the training set, the values ranges from 180 to 590. Two new values are added to improve the diversity of the dataset. For each case, we train on 4 Reynolds number and test on the remaining one that was left out. This has the purpose to verify if our model can generalize on unseen data points. Case 1 and 3 are outside of the training set range and we expect both case to have lower accuracy score.

Case	Training set	Test set
1	Re_τ = [285, 395, 450, 590]	Re_τ=180
2	Re_τ = [180, 285, 450, 590]	Re_τ=395
3	Re_τ = [180, 285, 395, 450]	Re_τ=590

Table 4.1. Three training-prediction cases

The training stops when both the validation loss starts to rise and the patience argument is invoked. In machine learning, increase of validation loss is a strong indicator for over-fitting, since the model is incapable of giving the correct output on unseen data. The model's epoch ranged between 2000-4000. The Figure below shows an example of training and validation loss as a function of epochs during a training of 100-Neurons-MLP for case 1. The R^2 score is a statistical measure that determines how well a regression prediction approximates the true data points. An R^2 of 1 indicates a perfect fit. Comparing across the three cases from the Table 4.2, we notice Case 2 has considerably better accuracy. One explanation is that Case 1 and 3 testing data is outside of the training range. Furthermore, the reason Case 3 has the worst outcome might stem from the fact the training dataset yields low-Reynolds number effects. Thus, a solution would be to include training data of higher Reynolds to balance the effect of lower and higher Reynolds number.

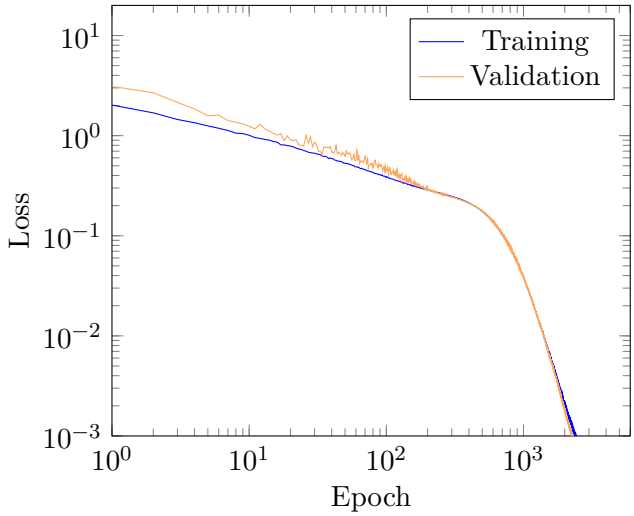

Figure 4.23. Learning Curve - TC

The Figure 4.23 shows an example learning curve of Case 1. The plot starts with signs of over-fitting but with time the both the training and validation learning curve merges. However, they both did not reach the point of stability since both learning curves continue to decrease. This is because the early-stopping criterion has detected the validation curve started to rise relative to the training loss. Which leads to over-fit and terminated the training process.

4.1.3 Results

Since the turbulent flow is symmetric above and below the channel centreline. It was decided to train on the lower half of the channel data.

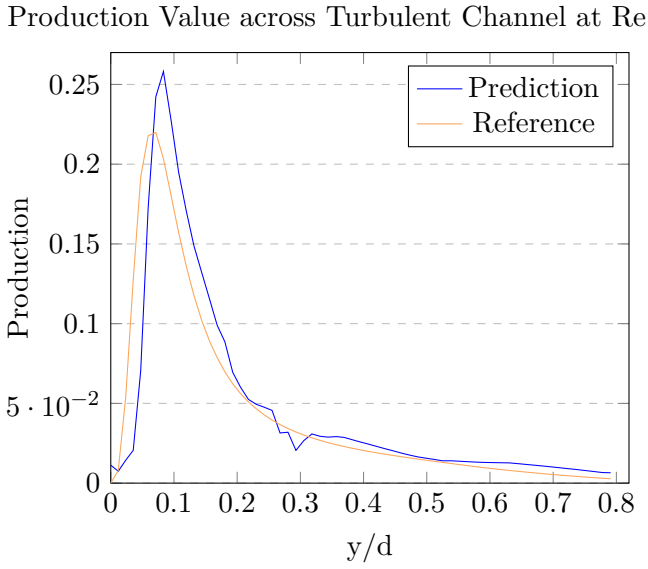

Figure 4.24. Production - Case 1, TC

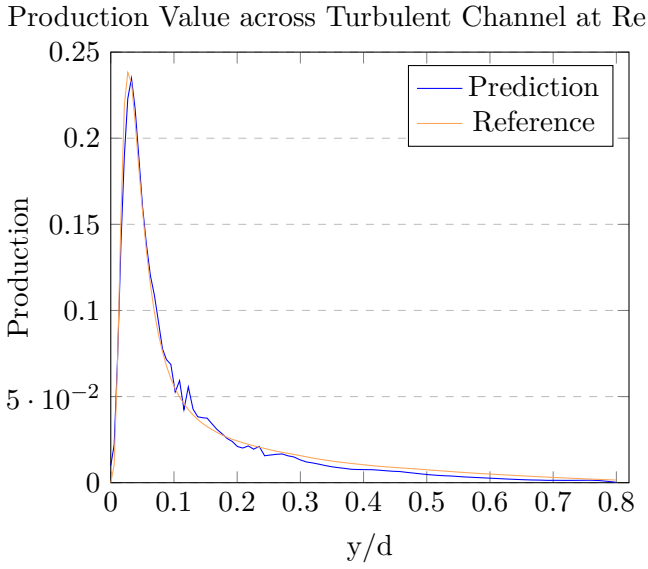

Figure 4.25. Production - Case 2, TC

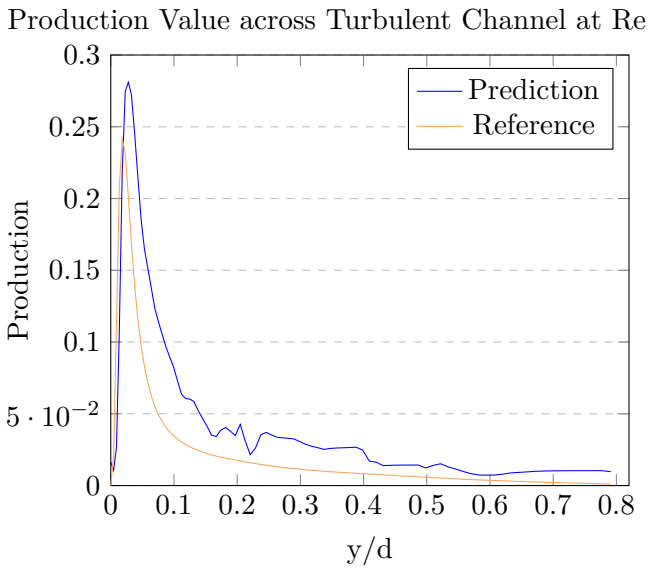

Figure 4.26. Production - Case 3, TC

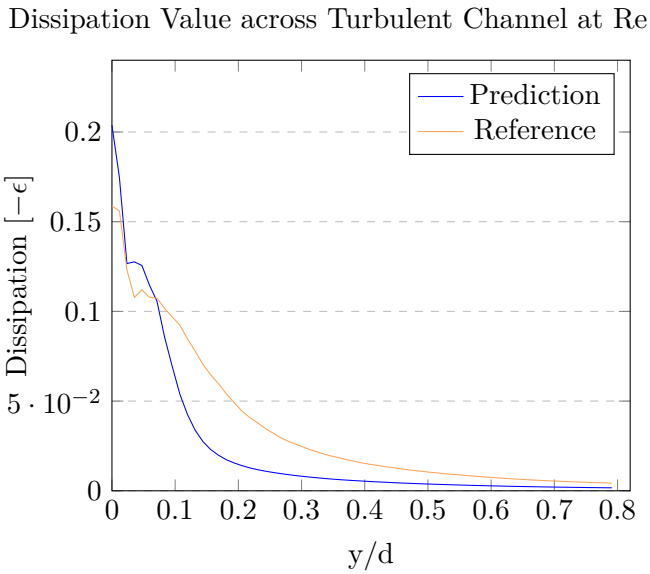

Figure 4.27. Dissipation - Case 1, TC

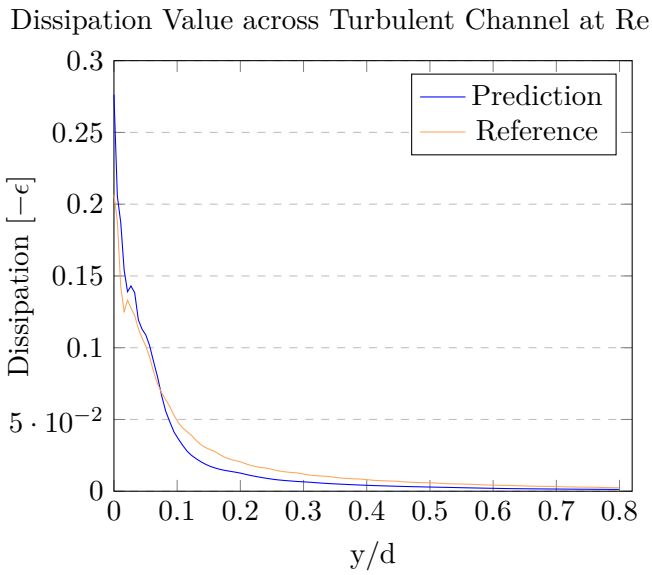

Figure 4.28. Dissipation - Case 2, TC

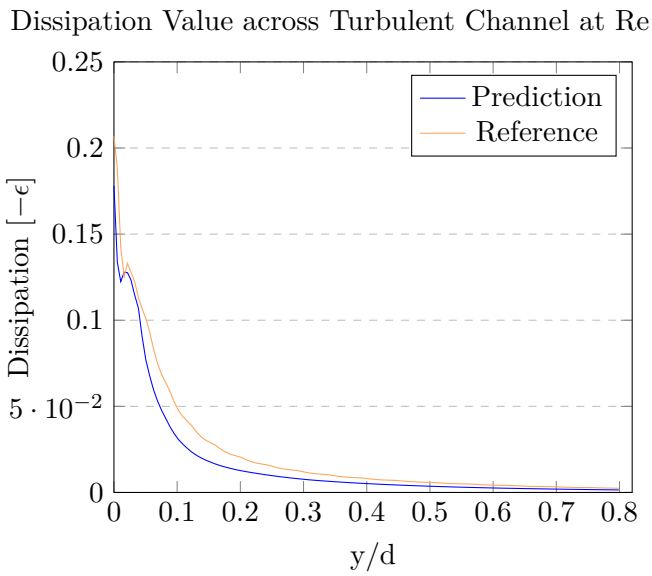

Figure 4.29. Dissipation - Case 3, TC

	Case 1		Case 2		Case 3	
	Train R^2	Test R^2	Train R^2	Test R^2	Train R^2	Test R^2
100-Neurons-Production	0.9178	0.8231	0.9921	0.9898	0.9217	0.7070
100-Neurons-Dissipation	0.9997	0.8228	0.9999	0.9572	0.9997	0.9140

Table 4.2. R^2 of production & dissipation predictions, TC

Regression training on the original dissipation data were poor due to small value, skewed distribution, and its monotonically decreasing nature. A solution was to pre-process the dissipation value and take the negative natural logarithmic. By applying the log transformation, the boundaries became easier to recognize for the model and it also did not treated the tail region of the dataset as outliers anymore. Following the pre-processing, we observe accurate results on Figure 4.27, 4.28, and 4.29 with the highest score on Case 2 at $R^2 = 0.9572$.
The production data distribution has 1 high peak located on the left region of the graph within the viscous region of the flow. No prior transformation was performed on the data. Both prediction for Case 1 and 3 overpredicted the production, shown by Figure 4.24, and 4.26. This is a consequence of the out of distribution dataset for testing the model. The best result were obtained by Case 2, shown in Figure 4.25. With a $R^2 = 0.9898$.
As expected, we see better convergence on Case 2 for both production and dissipation which has kept a good agreement with the reference data. This increase in performance can be attributed to the fact the test set is contained in the boundaries of the training set.

4.2 NACA 0012 Airfoil

DNS of flow over a NACA 0012 airfoil are performed at $Re = 5 \times 10^4$ and $M = 0.2$. The Angle Of Attack (AOA) are from 4 to 12 degrees. The three-dimensional turbulent production and dissipation values are investigated. Finally, a machine learning model is built to predict the turbulent production and dissipation values of testing data at different angles.

4.2.1 Case Description

The geometry of an airfoil is defined by the camber, thickness ratio, and leading edge radius. From a practical standpoint, we want to extract lift, drag, and moments from the airfoil. Thus, we are interested in the pressure distribution. At low Reynolds numbers and at small AOA, the

flow on the suction side of the airfoil remains laminar. Conversely, a separation region will grow downstream due to a strong adverse pressure gradient at high angles of attack. The trailing edge turbulent region will move upstream with increasing angle of attack. Prediction of this flow transition and separation is difficult to capture using RANS model.

The first solution point off the wall is located at $y^+ = 0.5$. According to experimental data, it is within the viscous sublayer of a channel flow [**davidson2004turbulence_viscoussublayer**]. Each simulation is solved using $P = 3$ and run for 30 convective times. Statistical quantities are computed over 25 convective times. Additionally, the pressure coefficient (Cp) is generated at three sample angles to validate accuracy against reference data. Namely at 5, 8, and 12 degrees AOA. In Figure 4.35, the flow is transitioning from laminar to turbulent at around 20% of the chord. The Cp curve drops because the flow transitions to turbulence, driving a pressure recovery regime. This region tends to get widened with lower-order method, or with coarse meshes, because numerical dissipation effectively damps the transition mechanism. Therefore, using high-order methods with PyFR, we were able to generate more accurate results.

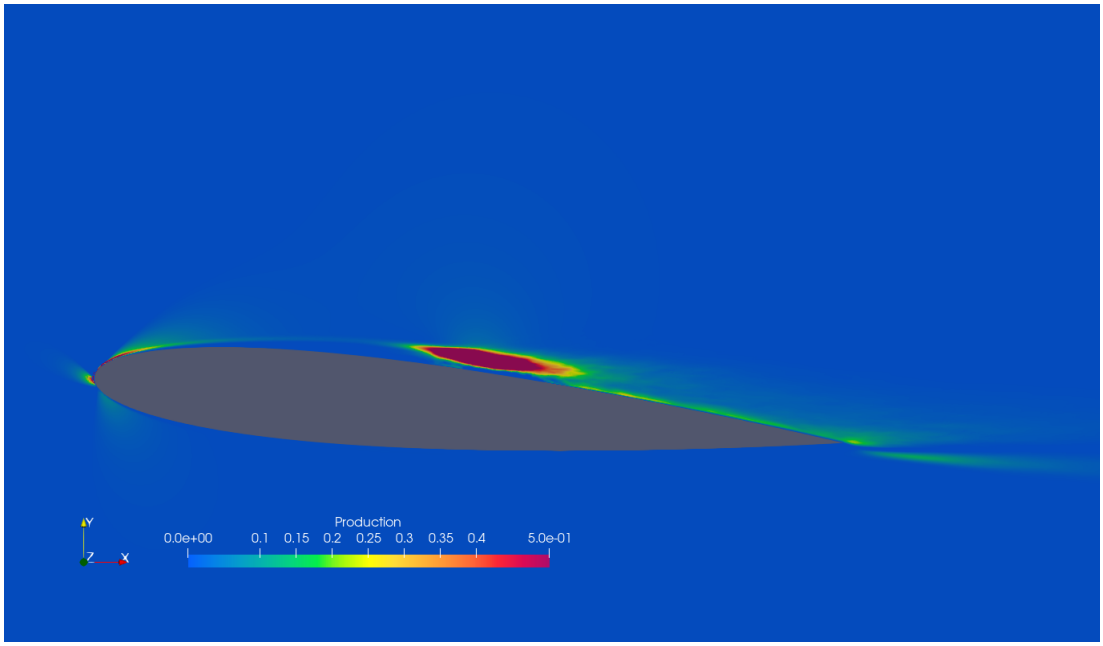

Figure 4.30. Turbulent kinetic energy production distribution at 5 degrees AOA

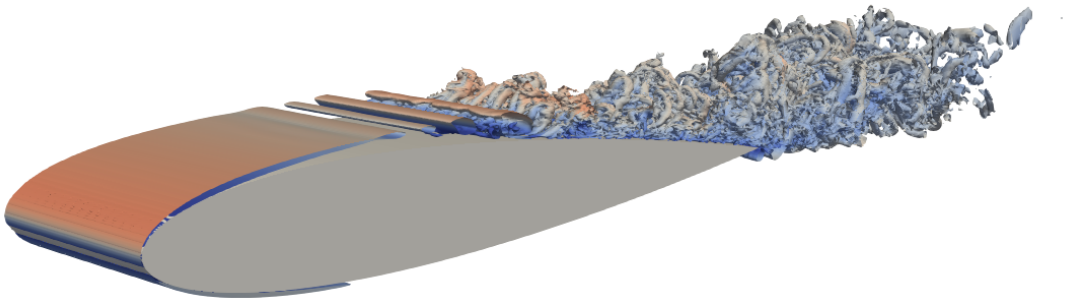

Figure 4.31. Q-criterion at 5 degrees AOA

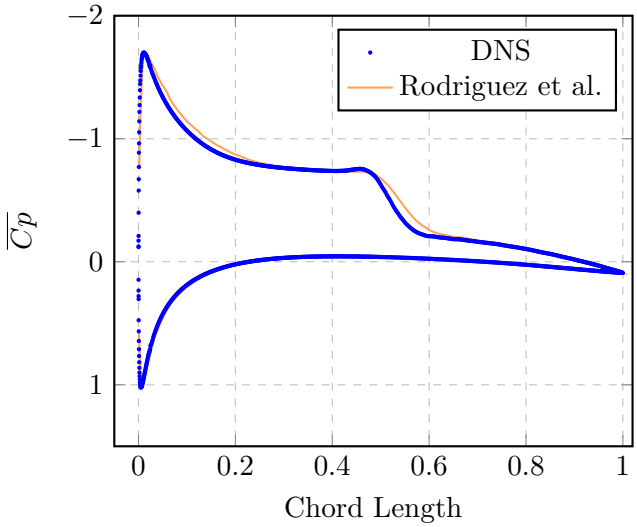

Figure 4.32. Time-averaged pressure coefficient at 5 degrees AOA

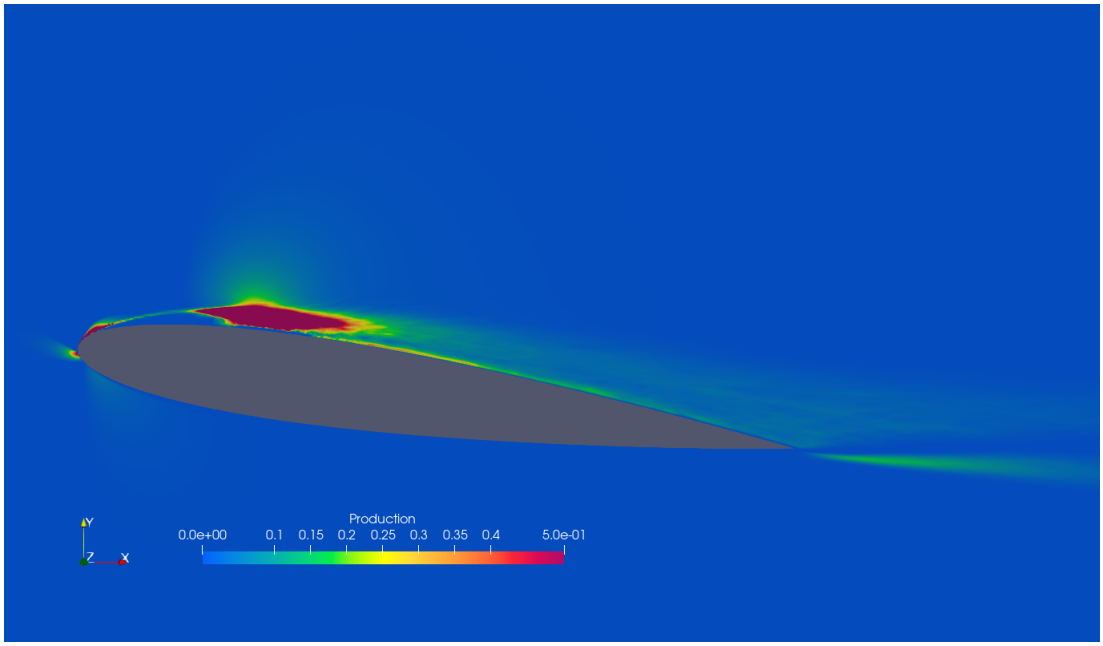

Figure 4.33. Turbulent kinetic energy production distribution at 8 degrees AOA

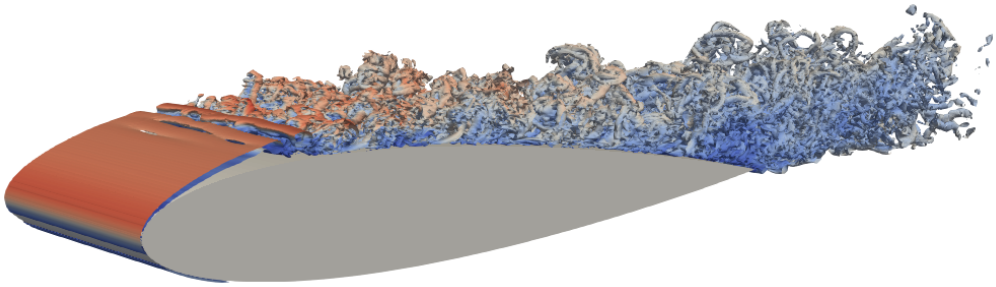

Figure 4.34. Q-criterion at 8 degrees AOA

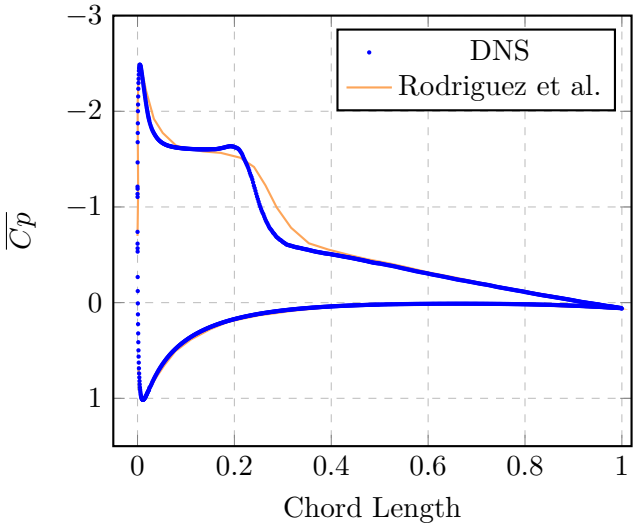

Figure 4.35. Time-averaged pressure coefficient at 8 degrees AOA

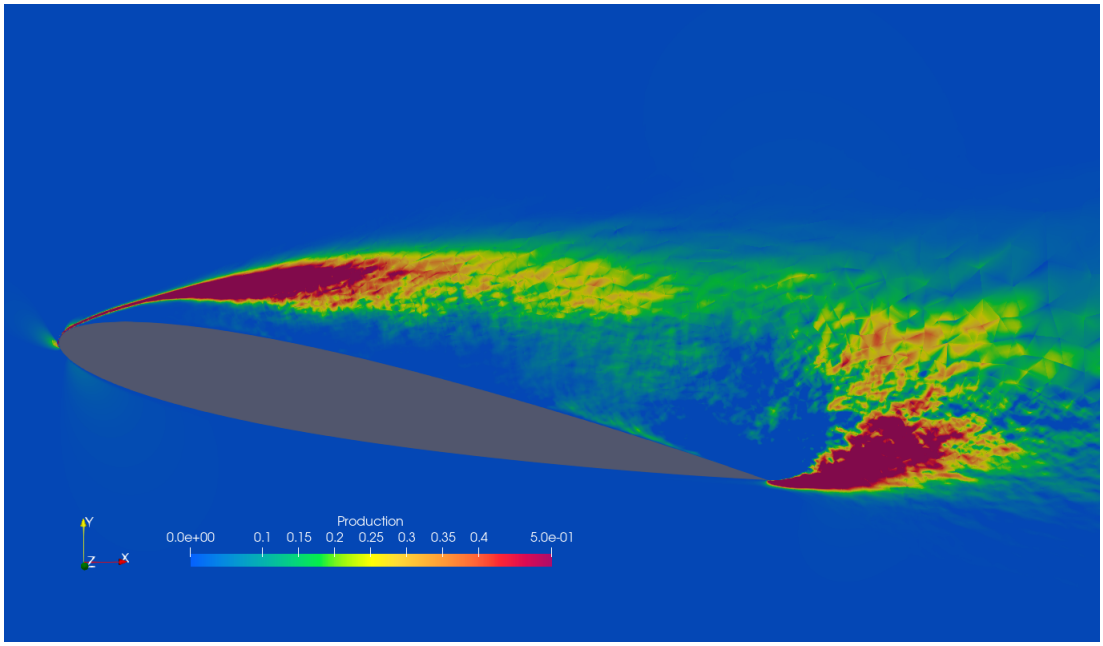

Figure 4.36. Turbulent kinetic energy production distribution at 12 degrees AOA

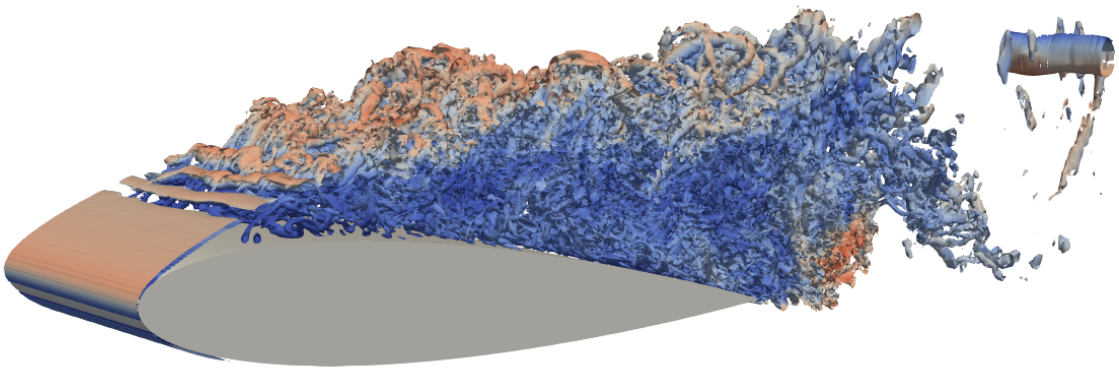

Figure 4.37. Q-criterion at 12 degrees AOA

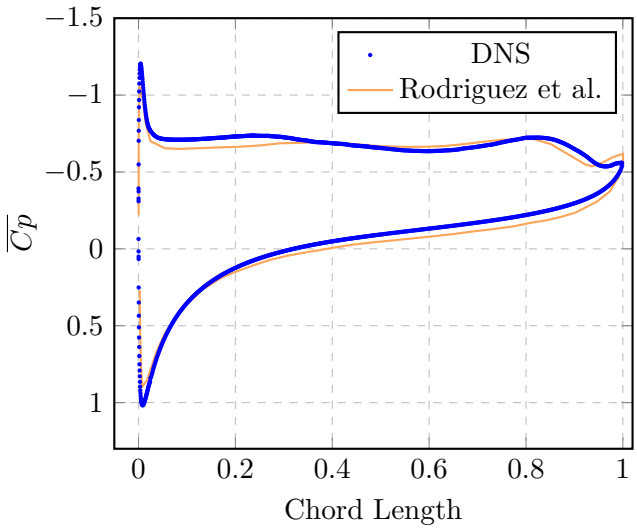

Figure 4.38. Time-averaged pressure coefficient at 12 degrees AOA

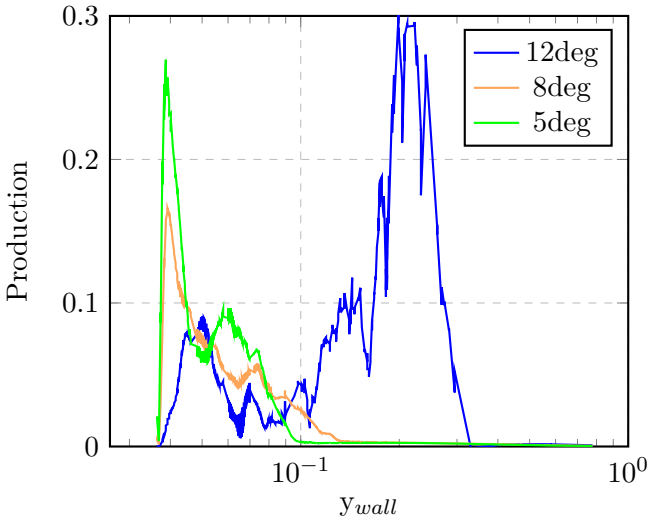

Figure 4.39. Turbulent Production at 0.7c

The production value for three sample degrees have been evaluated at 70% of the chord. From Figure 4.39, we notice for each AOA there are 2 peaks for production. These are attributed to the wall effects and the shear layer. We interpret those two zone as the largest concentration of cross-flow fluctuations which are growing in tandem with the TKE production value. The flow at 12 degrees is fully detached which explains the production peaks away from the wall. A fully detached flow at 12 degrees AOA behaviour can be visualized in Figure 4.42.

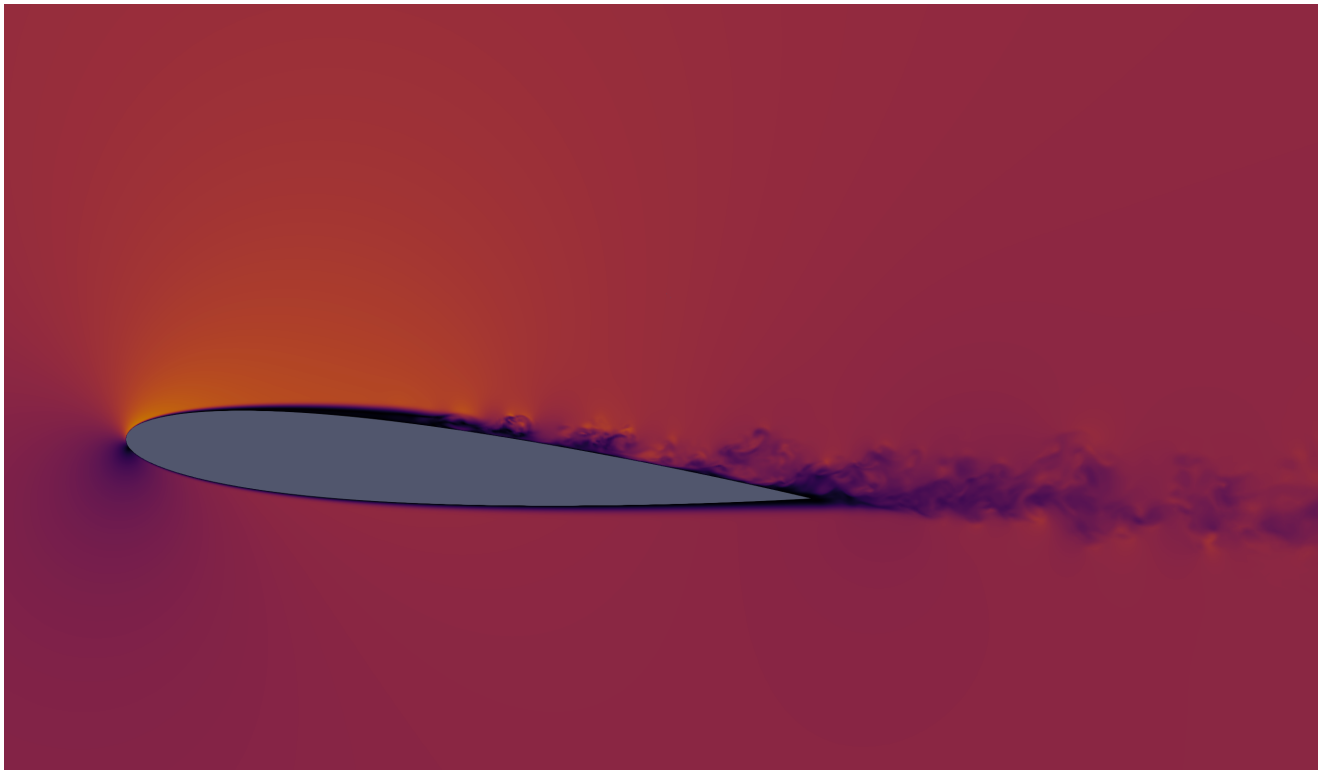

Figure 4.40. NACA 0012 DNS at 5 degrees AOA

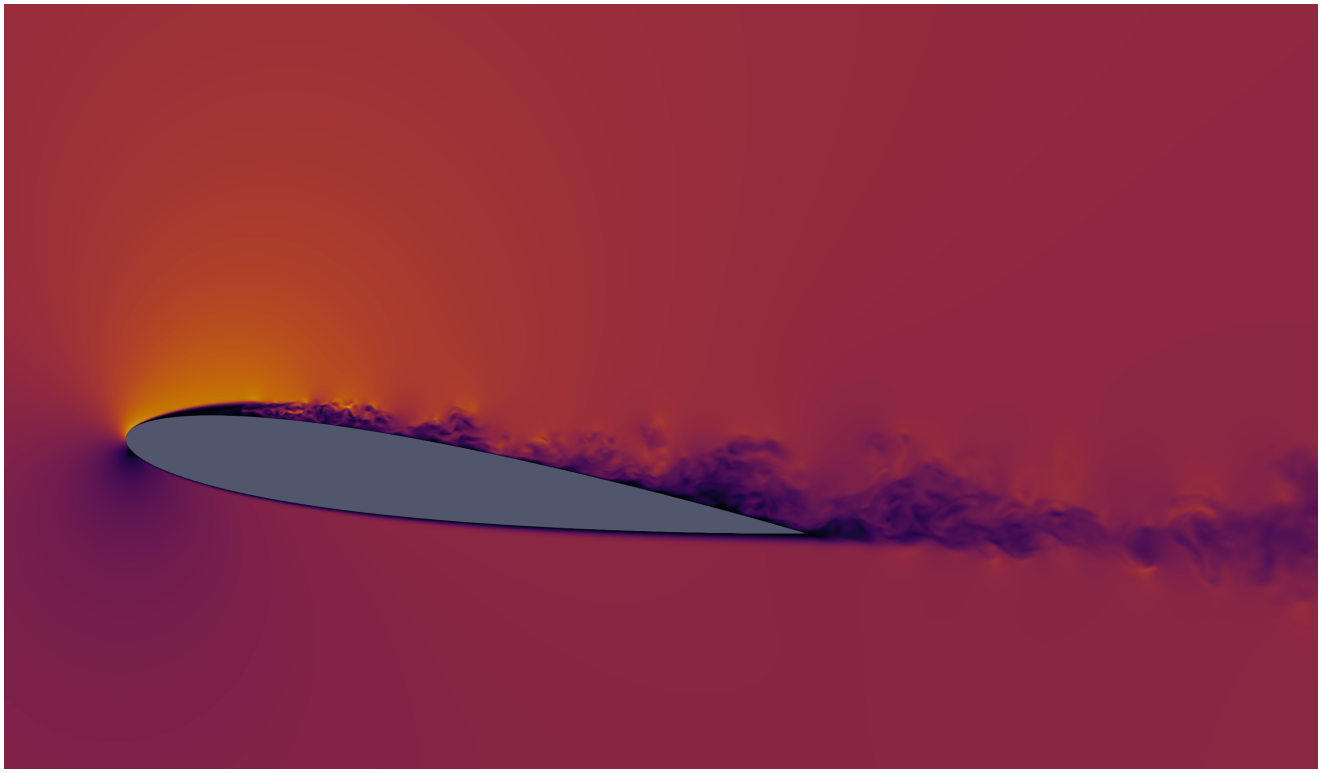

Figure 4.41. NACA 0012 DNS at 8 degrees AOA

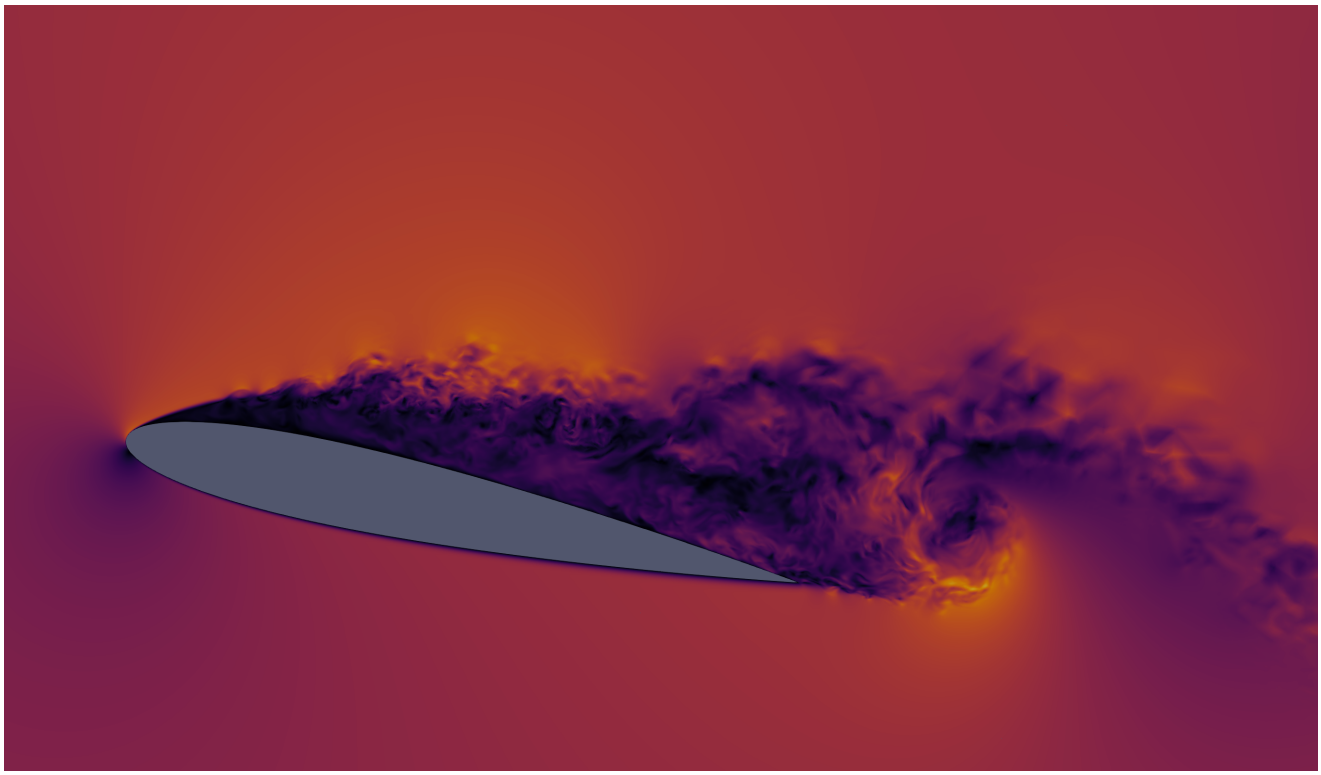

Figure 4.42. NACA 0012 DNS at 12 degrees AOA

4.2.2 Machine Learning Framework

Inspired by the success of the first experiment in Section 4.1.2, we take a similar approach for the NACA 0012 at Mach = 0.2. However, there are some noticeable differences between the first and second experiment. Namely, the NACA 0012 DNS is a quasi-two-dimensional flow, and generates significantly more data. Knowing this, we introduce two new features during the model training. They are the Euclidean wall distance and the turbulent kinetic energy. The intent is to give the model a spatial intuition of its surrounding coupled with the local amount of turbulent kinetic energy. This could inform the model about the flow properties and ultimately better generalize the kinetic energy production and dissipation values.

$$TKE = \frac{1}{2}\left(\overline{(u')^2} + \overline{(v')^2} + \overline{(w')^2}\right). \tag{4.5}$$

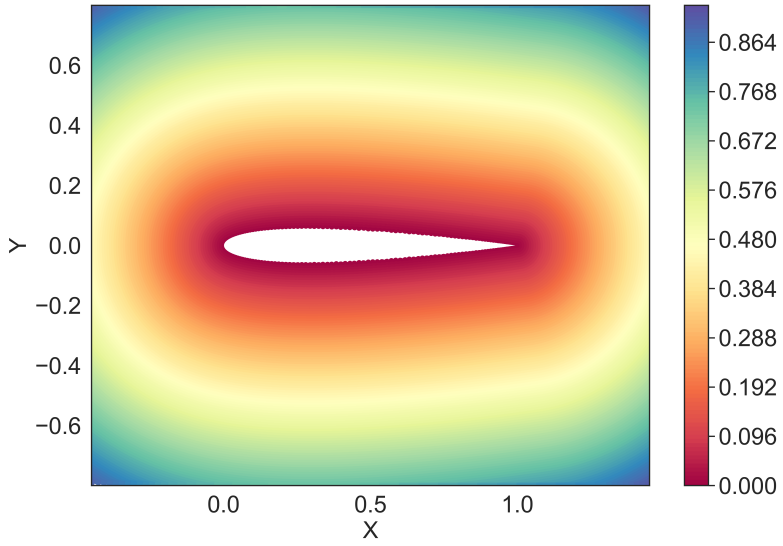

Figure 4.43. Wall distance contour, y_{wall}

In the case description section we showed that our simulation was able to generate reliable DNS data. We leverage this ability to simulate the dataset for AOA between 4 and 12. Allowing us to train the model and predict on unseen data. For each case, we randomly split the training data into 80% training and 20% validation data. The loss function used is the Mean Squared Error (MSE) between the predicted production or dissipation and the DNS value. The Adam optimization and ReLU activation are used with an initial learning rate of 5 x 10^{-7} and a batch size of 10. The weights of the networks were initialized with He uniform and biases were

initialized as zeros. The neural network architecture is 5 layers deep with 64 units on each layer. Batch normalization is added to accelerate the training. The features are

$$x,\ y,\ y_{wall},\ \overline{\rho},\ \overline{\rho u_i},\ \overline{\rho E},\ TKE,\ \overline{\frac{\partial \rho u_i}{\partial x_i}}. \tag{4.6}$$

From the turbulent channel case, we have learned that the generalization perform best when the testing set is not situated out of the training boundaries. Therefore, Case 1 and 2 have their test set fixed in the middle of the training set. However, on Case 3 we fixed the test set to be outside of the training range. We expect to have better result in Case 1 and 2 over Case 3. This is summarized in Table 4.3.

Case	Training set	Test set
1	AOA = [4°, 6°]	AOA=5°
2	AOA = [7°, 9°]	AOA=8°
3	AOA = [10°, 11°]	AOA=12°

Table 4.3. Three training-prediction cases

Figure 4.44 shows the learning curve of Case 1. The validation curve is under the training meaning it has not over-fitted the training data. Early stopping ended the training after 319 epochs due to no validation loss improvement.

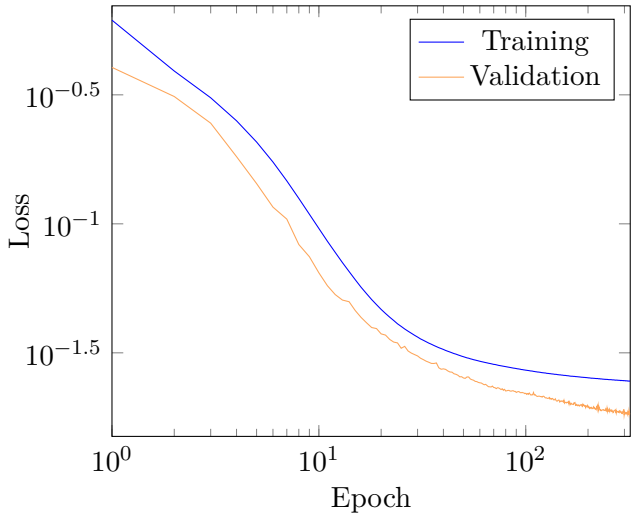

Figure 4.44. Loss vs epochs for NACA0012 5° AOA

4.2.3 Results

In this section we present the outcome of training a neural network on DNS to predict Production and Dissipation values at 0.7 chord.

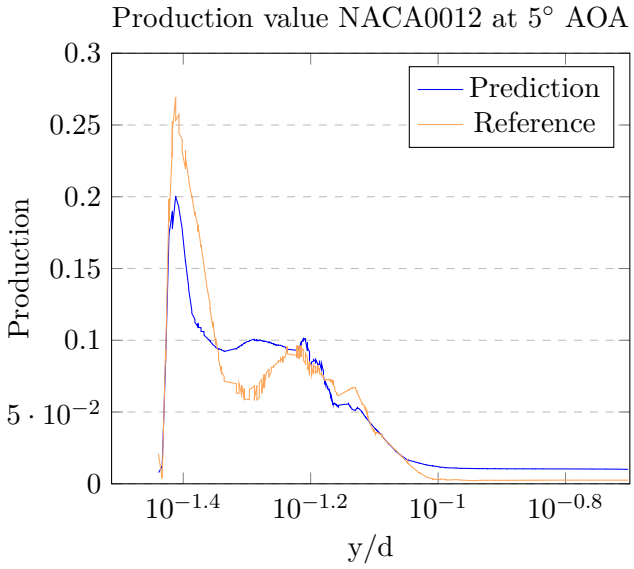

Figure 4.45. Production - Case 1, NACA0012

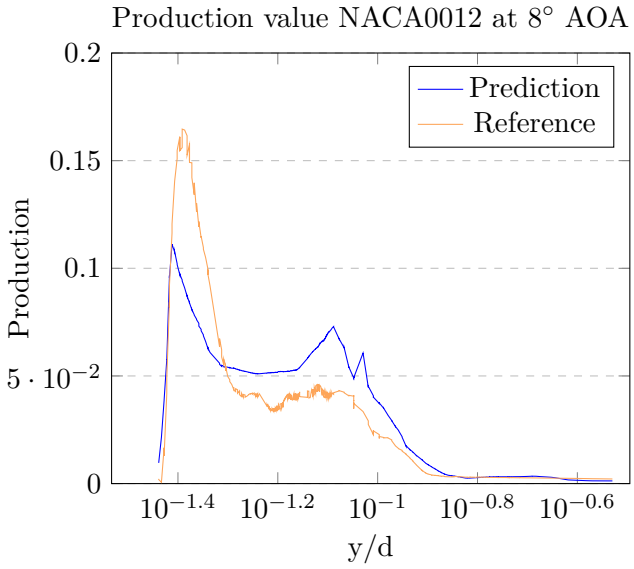

Figure 4.46. Production - Case 2, NACA0012

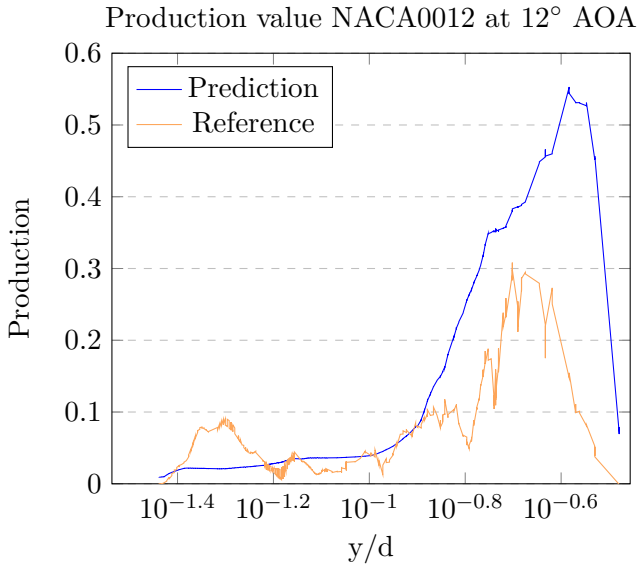

Figure 4.47. Production - Case 3, NACA0012

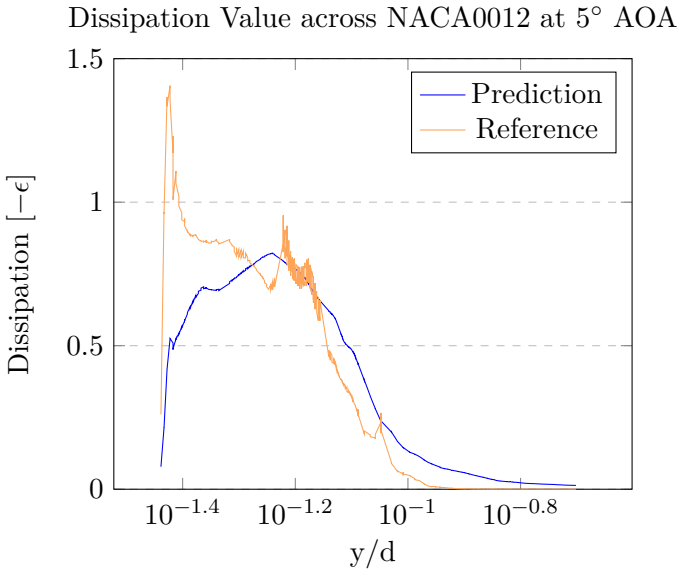

Figure 4.48. Dissipation Case 1, NACA0012

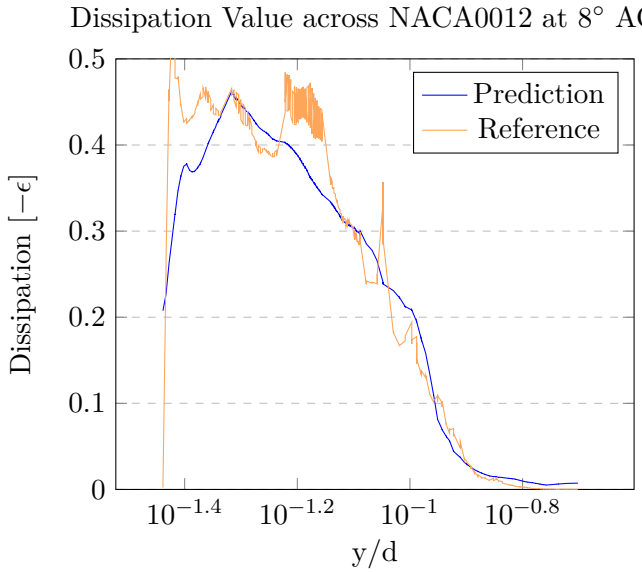

Figure 4.49. Dissipation Case 2, NACA0012

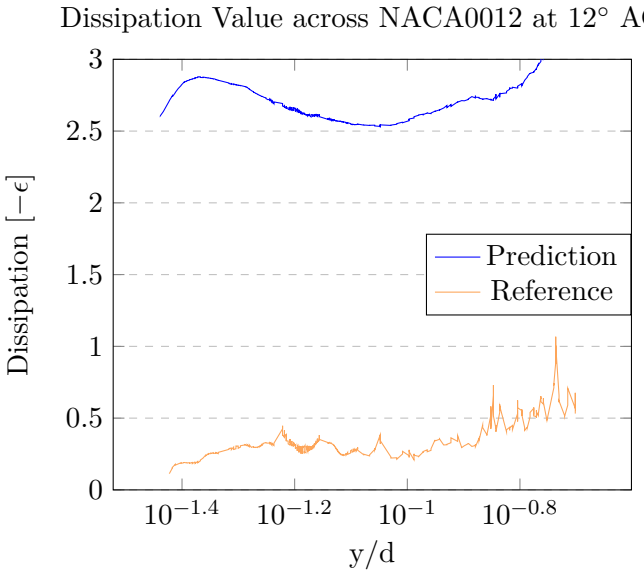

Figure 4.50. Dissipation Case 3, NACA0012

	Case 1	Case 2	Case 3
	Test R^2	Test R^2	Test R^2
64-Neurons-Production	0.7792	0.5277	0.4261
64-Neurons-Dissipation	0.7203	0.8931	0.0019

Table 4.4. R^2 of production and dissipation predictions, NACA0012

Training a model where the test set is bounded within the training region results in good

agreement with the reference data. As the AOA increases, the flow over the airfoil starts to detach and alters the behaviour of the production and dissipation distribution. In the context of predicting production values. The model was successfully able to predict the bi-modal behaviour of the reference data at their peak locations. As illustrated by Case 2 Figure 4.49, prediction is accurate due to the absence of a sharp dissipation peak, unlike case 1. Instead, Case 2 exhibits a more smooth dissipation transition away from the wall. However, the production and dissipation for Case 3 displays an out-of-distribution generalization. Resulting in low prediction accuracy. This outcome is expected because the model lacks the robustness to evaluate the chaotic nature of the test data that are different from the training set. From Figure 4.39, we observe the flow patterns at different AOA. The production peak shifts from left to right which in turn illustrates how Case 3 does not have the training data to cover the region for the test AOA.

4.3 Discussion

While the results were not as accurate as the previous turbulent channel case. It is relevant to remember what was fed to the neural network. The feature list for the 3D airfoil case is given by Equation 4.2.2. Which does not cover all the requirement of production and dissipation - Equations 3.2.1 and 3.2.1. It was done so to avoid having cross correlations terms as features. As they are not readily available in the RANS formulation. Therefore, the neural network was able to find a mapping to predict the objectives without the full knowledge of the flow. This shows promise for further methodological developments that incorporate neural network in turbulence modeling. In summary, applying a feed forward neural network approach to find the turbulent values for production and dissipation proves to be a difficult task but if applied correctly can yield good results.

Chapter 5

Conclusion and Future Work

This study has shown the capabilities and limitations of MLP for turbulence modeling. Specifically, a data driven approach was taken to predict the turbulent kinetic energy *production* and *dissipation* values in the TKE transport equation. Training data were generated through a DNS for the turbulent channel and the NACA0012 in 3D. The DNS were performed using PyFR. Benchmarks were performed to evaluate the accuracy and the approach proved to have even better accuracy than the reference data. For instance the Cp plots on Figure 4.32, 4.35, 4.38 were compared to the reference data from Rodriguez, et al. [**NACA0012**]. This work shows the combination of high fidelity CFD and ML have the potential to make a truly meaningful contribution to enhance or replace turbulence models used with RANS.

The first experiment used the turbulent channel case to validate the methodology. Without being given the full set of variables, the network learned from DNS to have its own representation of the *production* and *dissipation*. Furthermore, it is able to predict on unseen data given it has been trained on physically similar configurations. The optimal model results matched with 99% R^2 on the turbulent channel case for the production dataset. With the lowest being 70% R^2 for an out of distribution generalization case. Similarly, on the dissipation dataset. The model achieved 82% R^2, 95% R^2, and 91% R^2 for Case 1 to 3.

Following the encouraging results from the validation experiment, a more challenging complex non-homogeneous flow case was tested with an MLP. Using a similar architecture, the predicted values reached good agreement on low AOA. The model achieved 77% R^2 for case 1 on the production dataset. While Case 2 and 3 achieved respectively 53% R^2 and 43% R^2. The better

performance on low AOA can largely be attributed to the the lack of flow separation between the different configurations. Similarly, on the dissipation dataset. The model achieved 72% R^2, 89% R^2, and 0% R^2 for Case 1 to 3.

5.1 Future Work

This research shows encouraging results, but several other directions are possible. A straightforward improvment would be of applying a rigorous hyperparameter exploration, since only a preliminary grid search was performed in this thesis. Physics-informed Neural Network (PINN) is the next step and should be explored to increase prediction accuracy. For instance, one can enforce the no-slip boundary condition at the wall. This is achieved by re parametrizing the output from the MLP to the corresponding value. This is motivated by the fact that the MLP has no embedded knowledge of any physical laws. Rui Wang et al. [**RuiPINN**] is another example of PINN. Using an autoencoder they have introduced the Turbulent-Flow Net. A hybrid approach that combines physical principles (RANS-LES coupling) and learned representation. The encoder is split into three modules in which each represents the turbulent flow at different scales. The decoder is a multi-layered deconvolutional layer that takes the aggregation of the previous three encoders and outputs a 2D velocity field. Currently, data fed to the MLP is only from local effects. However non-locality should be also addressed and added to the model. The argument is that the Reynolds stresses not only depend on the shape and intensity of local eddies (rate of strain). Instead it is affected by a chain of events from past eddies prior to arrival at the point of interests [**davidson2004turbulence**]. It is possible to model spatial non-locality using fractional calculus [**Songfractional**]. Finally, we examined how the eddy viscosity relates the Reynolds stress and rate of strain linearly through Equation 3.2. This is in fact incorrect for many flows since turbulence is inherently anisotropic. To consider anisotropy, a model should be trained to learn the Reynolds stresses directly, or infer them form the turbulent kinetic energy quantities. In the future, we can modify the RANS approach. The turbulence model would rely on the turbulent kinetic energy transport equation coupled with our neural network to predict the distribution of turbulent kinetic energy in the domain. Then, this distribution would be coupled with an equation to predict turbulent viscosity, similar to the equations used by two-equation turbulence models, such as k-epsilon.

Appendix A: Airfoil Coordinates Transformation

When building a neural network we need a reliable source of dataset for training our model. In addition to the freestream values and gradients we require the flow location. This is particularly important for quasi-two dimensional flow such as the NACA0012. To help generalize on unseen angle of attack, the distance between the point of interest and the closest wall location (airfoil) is calculated. This is possible by simply taking the Euclidean distance in a for loop and save the shortest distance. The NACA0012 coordinates are available online.

```
import numpy as np
import pandas as pd
import math

# airfoil coordinates and DNS simulation
df_airfoil = pd.read_csv(r"viz-airfoil.csv")
df_DNS = pd.read_csv(r"viz-DNS.csv")

def shortest_wall_distance(k):
    shortest_distance = []
    for i in  range (df_airfoil.shape[0]):
        p1 = [df_airfoil['x'][i], df_airfoil['y'][i]]
        p2 = [df_DNS["Points:0"][k], df_DNS["Points:1"][k]]
        distance = math.sqrt( ((p1[0]-p2[0])**2)+((p1[1]-p2[1])**2) )
        shortest_distance.append(distance)
# Take the shortest one and make y_wall = np.min(l)
    l = np.asarray(shortest_distance)
    return np.min(l)
```

Listing 5.1. Automate search for wall values

Appendix B: Neural Network with Tensorflow

```
def build_model(input_shape):
    """
    The weights are initialised by providing the input_shape argument in the
    first layer. Function should return the Sequential model.
    """
    model = tf.keras.Sequential([
        Dense(64, activation='relu', kernel_initializer='HeUniform',
              input_shape=input_shape, name='dense_one'),
        BatchNormalization(),
        Dense(64, activation='relu', kernel_initializer='HeUniform'),
        BatchNormalization(),
        Dense(64, activation='relu', kernel_initializer='HeUniform'),
        BatchNormalization(),
        Dense(64, activation='relu', kernel_initializer='HeUniform'),
        BatchNormalization(),
        Dense(64, activation='relu', kernel_initializer='HeUniform'),
        BatchNormalization(),
        Dense(1, kernel_initializer='HeUniform')])
    return model

def compile_model(model):
    """
    This function takes in the model returned from the build_model function,
    and compiles in-place with an optimiser, loss function and metric.
    """
    opt = keras.optimizers.Adam(learning_rate=5e-7)
    model.compile(loss='mae', optimizer=opt, metrics=[tf.keras.metrics.
    MeanAbsoluteError()])
```

```
26
27 def train_model(model, train_data, train_targets, epochs):
28     """
29     This function should train the model for the given number of epochs on the
       train_data and train_targets. Function returns the training history model.
       fit.
30     """
31     history = model.fit(train_data, train_targets, epochs=epochs,
       validation_split=0.2, batch_size=10, verbose=1,
32                         callbacks=[checkpoint, tf.keras.callbacks.EarlyStopping
       (monitor='val_loss', patience=20)])
33     return history
34
35 model = build_model(input_shape=(x_train_standard.shape[1],))
36
37 compile_model(model)
38
39 checkpoint_path = 'model_checkpoints' # if you save the whole model it will
40                                       # create a directory automatically.
41
42 checkpoint = ModelCheckpoint(filepath=checkpoint_path,
43                              save_weights_only=False,
44                              save_freq='epoch',
45                              monitor='val_loss',
46                              save_best_only=True,
47                              verbose=1)
48
49 ##########################
50 epochs = 500
51 ##########################
52 history = train_model(model, x_train, y_train, epochs=epochs)
```

Listing 5.2. Building a Sequential model