POLITECNICO DI TORINO

Faculty of Engineering
Master's degree in Aerospace Engineering

MASTER'S THESIS

Robustness-based training and explainability of a data-driven model to cure the inconsistency between RANS and DNS datasets

Supervisors

Prof.ssa Sandra PIERACCINI
Prof. Miguel Alfonso MENDEZ
Ing. Matilde FIORE

Candidate

Enrico SACCAGGI

APRIL 2023

Abstract

The present study proposes the use of Machine Learning to model turbulent heat transfer in liquid metal cooled nuclear reactors. Traditional modelling methods are found to have limitations, and the availability of high-fidelity data for low Prandtl number fluids motivates the use of advanced regression techniques.

The work is based on the analysis and the extension of the data-driven model developed by M. Fiore [1] to simulate liquid metal heat transfer which is described in subsection 2.2.1. The original Neural Network model was trained with high-fidelity data only. This approach was found to be limiting when the model is coupled with common momentum Reynolds Averaged Navier Stokes (RANS) closures due to the inconsistency between Direct Numerical Simulation (DNS) and RANS turbulence input data. In particular, the accuracy of the data-driven model dramatically decays when the network is applied in combination with momentum models based on the Boussinesq hypothesis e.g. the $k - \epsilon$ turbulence model.

In chapter 3 datasets obtained from RANS simulations and DNS for the same flow conditions were compared and dimensional reduced with a Principal Component Analysis (PCA) algorithm. An artificial neural network model for turbulent heat flux prediction was trained using the PyTorch framework. The accuracy of the predictions is evaluated through a loss function that considers the results from both input datasets, with the Pareto front constructed from multiple training runs (chapter 4). This analysis showed the possibility to train a network that is able to identify the nature of the input database and gave reasonable predictions with both sets of input data. In chapter 5 an *a priori* and *a posteriori* validation were carried out to test the model's performances. In particular, the newly trained network was implemented in OpenFoam which is a Computational Fluid Dynamic (CFD) software.

Finally, an interpretability analysis with the Shapley values algorithm was performed in chapter 6 to understand the peculiarity of the models trained with hybrid DNS-RANS input datasets.

Acknowledgements

Desidero esprimere la mia gratitudine a colei che ha reso possibile quest'esperienza di crescita accademica e personale: la Prof.ssa Pieraccini. Senza di lei non sarebbe stato possibile arrivare in Belgio e specializzarsi in questo ambito, che spero di indagare ancora in futuro.

Un ringraziamento speciale va al Prof. Mendez e a Matilde Fiore per il loro prezioso e costante aiuto durante la realizzazione di questo documento, in particolar modo per la loro pazienza, disponibilità e i loro insegnamenti, fondamentali per la mia formazione professionale.

Ringrazio sinceramente anche i miei genitori, Andrea e Barbara, i quali sono sempre rimasti al mio fianco sostenendo ogni mia scelta accademica e personale.

Ci tengo a ringraziare anche tutti i miei famigliari, iniziando dai miei nonni Agnese, Franco, Liliana e Luigi che mi hanno cresciuto con affetto rendendomi la persona che sono oggi; le mie cugine Francesca ed Ilaria che con gioia hanno alleggerito questo percorso a volte difficile; mio zio Amos e le zie Lara e Monica che hanno sempre creduto in me fin da quando ero bambino.

Durante questo percorso é stato fondamentale l'Asse Trapani-Beaulard: Linda, Miki e $\mathcal{T}$au, che ringrazio per le serate insieme, le pizzate della domenica sera, le cene in sessione e soprattutto per avermi fatto sentire a casa in una nuova città.

Grazie anche a tutti i miei compagni di corso, i miei amici di Reggio Emilia, di Torino, della Compagnia ad Alta Quota ed i colleghi di EoliTo che hanno reso possibili viaggi, divertimenti ed esperienze uniche. Un ringraziamento anche alle persone conosciute al von Karman ed ai miei coinquilini belgi per aver creato un ambiente amichevole e per le lezioni di francese.

Ringrazio Laura per avermi supportato e sopportato, per aver incoraggiato qualsiasi scelta presa, e soprattutto per essere sempre stata al mio fianco nonostante la distanza.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Machine Learning for Fluid Mechanics and Turbulent Flows

The field of fluid mechanics is in a state of rapid evolution, led by unprecedented volumes of data from experiments, field measurements, and large-scale simulations at multiple spatiotemporal scales. Machine learning provides a set of tools to extract information from data that can be translated into knowledge. Moreover, machine learning algorithms can augment domain knowledge and automate tasks related to flow control and optimization [2].

In this chapter, the current techniques and the emerging opportunities of machine learning in fluid mechanics are presented. At the end of this chapter, a short description of turbulent flows will be done.

1.1 Introduction

The High-Performance Computing (HPC) architecture made it possible to work with large amounts of data. Over the last 50 years, many techniques were developed to handle such data, ranging from advanced algorithms for data processing and compression to fluid mechanics databases.

According to [2] can be observed a trend in :

- A vast and increasing volumes of data;
- Advances in computational hardware and reduced costs for computation, data storage, and transfer;
- Sophisticated algorithms;

- A growth of open source software;
- Big investments from industry on data-driven problem-solving.

All of these advances have fueled interest in machine learning to extract information from this huge amount of data. These learning algorithms may be categorized into supervised, unsupervised and semisupervised learning as shown in Figure 1.1.

Supervised learning is defined as learning from data labeled with expert knowledge by providing correct information to the algorithm; unsupervised learning is defined as learning without labeled data and semisupervised learning is defined as learning with partially labeled data or by interaction of the machine with the environment [2].

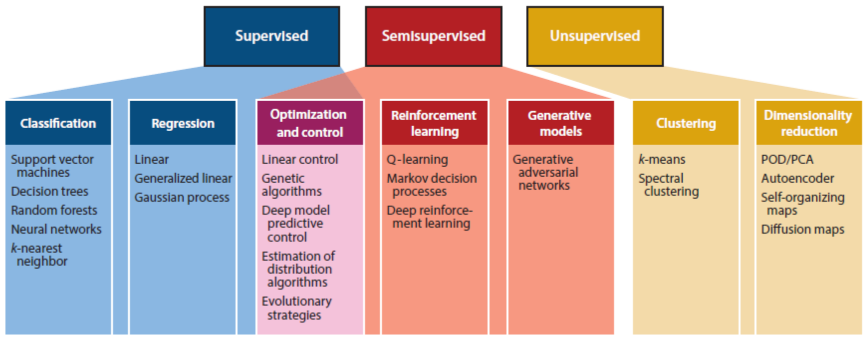


Figure 1.1: Categorization of the machine learning algorithms [2].

1.2 Historical overview

The interface between machine learning and fluid dynamics has a long history. Kolmogorov, a founder of statistical learning, in the early 1940s, considered turbulence as one of the prime applications of machine learning.

Ingo Rechenberg and Hans-Paul Schwefel at the Technical University of Berlin in the 1960s performed experiments in a wind tunnel on a corrugated structure composed of five linked plates with the goal to find the optimal angle in order to reduce the drag [2].

The interest in machine learning and neural network, which is a computational architecture based loosely on a biological network of neurons used for non-linear problems, increased with the development of the backpropagation algorithm. This one allows the training of neural networks with multiple layers. In the 1990s a lot of work using neural networks was made in the context of trajectory analysis and classification.

The link between proper orthogonal decomposition (POD) and linear neural networks was exploited in order to reconstruct turbulence flow fields and the flow in the near wall region of a channel flow starting from DNS results [3]. Tracey et al. [4] used machine learning algorithms to model the Reynolds stress anisotropy tensor and also to mimic the source term of the Spalart-Allmaras model. Duraisamy et al. [5] used neural networks and Gaussian processes to model intermittency in transitional turbulence and Zhang et al. [6] used these algorithms to model turbulence production on a channel.

1.3 Machine Learning fundamentals

"Learning is the process that aims to estimate an unknown (input, output) dependency or structure of a system using a limited number of observations" (V.Cherkassk) in [7]. The general learning scenario involves three components: a generator, a system, and a learning machine as shown in Figure 1.2.

- The Generator produces random vectors $\vec{x}$ drawn independently from a random probability density $p(\vec{x})$ which is unknown;

- The System produces an output value $\vec{y}$ for every input $\vec{x}$ according to the fixed conditional probability density function $p(\vec{y} \mid \vec{x})$ which is also unknown;

- The Learning Machine (LM) is able to implement a set of functions $\Phi(\vec{x}, \vec{y}, \vec{w})$ where $\vec{w}$ are a set of abstract parameters (called weights in the neural network framework) of the LM that need to be found in the training process [7].

The learning process of the machine can be summarized as the minimization of risk functional:

$$R(\vec{w}) = \int L\left[\vec{y}, \Phi(\vec{x}, \vec{y}, \vec{w})\right] p(\vec{x}, \vec{y}) d\vec{x} d\vec{y} \tag{1.1}$$

where $\vec{x}$ are the inputs; $\vec{y}$ are the samples from a probability distribution p; $\Phi(\vec{x}, \vec{y}, \vec{w})$ define the structure of the learning problem; $\vec{w}$ represent the parameters of the LM and L the loss function that balance the various learning objectives (e.g. accuracy and simplicity).

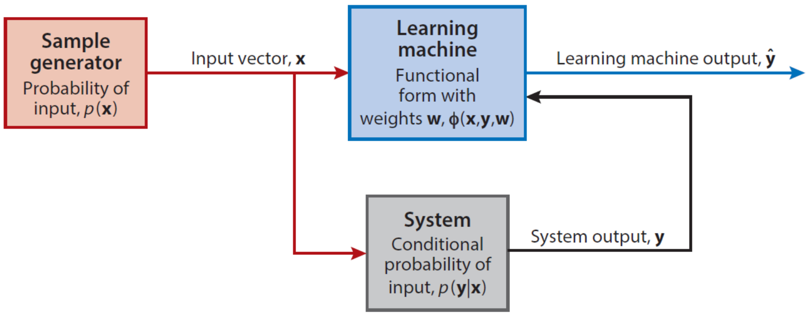

Sample generator
Probability of input, p(x)
Input vector, x
Learning machine
Functional form with weights w, ϕ(x,y,w)
Learning machine output, ŷ
System
Conditional probability of input, p(y|x)
System output, y

Figure 1.2: The Learning problem: a learning machine uses inputs generated from the sample generator and the observation from the system to approximate the inputs [2].

The learning problem can be split into three categories as it can be noted in Figure 1.1: supervised, semisupervised, and unsupervised learning. The next sections will take a closer look at these categories.

1.4 Supervised Learning

In supervised learning, models are trained with labeled datasets and the model learns about each type of data. The minimization of the cost function, which depends on the training data, will determine the unknown parameters of the LM. A common loss function can be defined as:

$$L\left[\vec{y}, \Phi(\vec{x}, \vec{y}, \vec{w})\right] = |\vec{y} - \Phi(\vec{x}, \vec{y}, \vec{w})|^2 \tag{1.2}$$

1.4.1 Neural Network

Neural networks are probably the most well-known methods in supervised learning. They are fundamental nonlinear function approximators, indeed following the universal approximation theorem any function can be approximated by a such deep neural network.

The key properties of these networks are their flexibility and modularity. They are composed of neurons and each neuron receives an input, processes it through an activation function, and produces an output. Multiple neurons can be combined into different structures that reflect knowledge about the problem and the type of data [2].

Neural network architectures have an input layer that receives the data, one or more hidden layers, and an output layer that produces a prediction. Each node, or artificial neuron, is connected to another and has an associated weight and threshold. The node is activated and can send data to the next layer if the output of a neuron is above the defined threshold. Otherwise, no data is passed along to the next layer of the neuron [8].

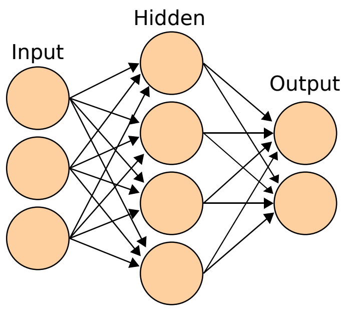


Figure 1.3: Example of a neural network with one hidden layer.

Each individual node can be seen as its own linear regression model, composed of input data, output data, weights, and a bias (threshold) the formula would look like this:

$$z = \sum_{i=1}^{m} w_i x_i + \text{bias} = w_1 x_1 + w_2 x_2 + w_3 x_3 + \text{bias} \tag{1.3}$$

In Equation 1.3 it's clear that these weights help determine the importance of any given variable, larger weights contribute more, lower weights less. The result of Equation 1.3 is then passed through an activation function which determines the output. An example of what has just been stated is shown in Figure 1.4

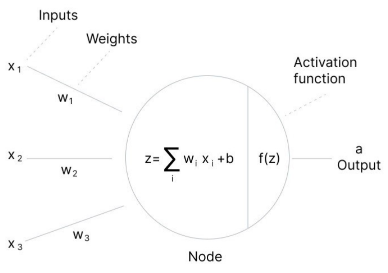


Figure 1.4: At the left the input of the selected neuron, the weights are represented as the connection and the activation function as $f(z)$ [9]

It's now important to define two essential terms: feedforward propagation and backpropagation. Feedforward propagation means that the flow of information occurs in the forward direction, starting from the input layer then to the hidden layers, and at the end to the output layer. Backpropagation is the action taken by the algorithms to repeatedly adjust the weights and biases in order to minimize the difference between the prediction and the real value (cost function). To minimize the cost function, the algorithm must compute the gradient referring to each weight in order to see which of these weights actually makes it change the most.

The purpose of the activation function is to add non-linearity to the neural network. The activation function introduces an additional step at each layer, although the computation time increases it is still worth it In fact, let's suppose that a neural network is without an activation function. In this case, the neurons will only perform a linear transformation on the inputs. It doesn't matter how many hidden layers are present, because all layers will behave the same way, since a composition of linear functions is also a linear function.

There are many activation functions but for simplicity's sake, are listed only three of them: the binary step, the sigmoid function, and the ReLU function.

Binary Step Function

The binary activation function is based on a threshold value and it is a linear activation function. The neuron is activated if the input is greater than a specified threshold value or else it gets deactivated. The binary step function is the simplest activation function that exists and can be implemented with only an "*if*" statement a can be seen in Equation 1.4 [10]. While creating a binary classifier a binary activation function is used but on the contrary, this type of function cannot be used in the case of multiclass classification. Another problem of this type of activation function is that the gradient is zero which can cause hindrance in the backpropagation. [10].

The function can be written as:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \tag{1.4}$$

Sigmoid/Logistic Activation Function

The sigmoid is a non-linear activation function that can allow using the backpropagation algorithm because its derivative is now related to the inputs and it can allow an understanding of which weights in the input neurons bring the best prediction. The sigmoid function transforms the values in the range 0 to 1. It can be defined as [9]:

$$\delta(x) = \frac{1}{1 + e^{-x}} \tag{1.5}$$

The sigmoid or the logistic function according to [9] is one of the most widely used function because :

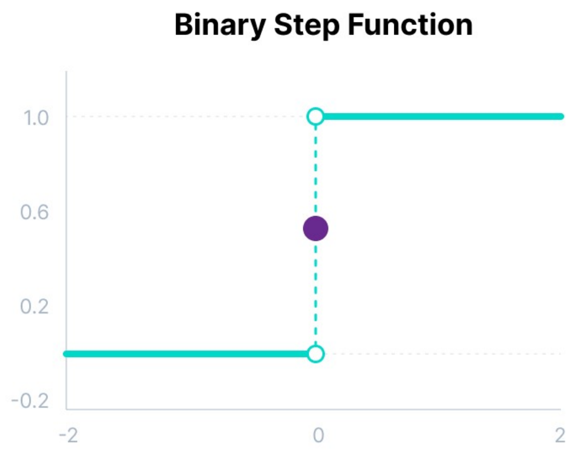


Figure 1.5: Representation of the binary activation function [9].

- It is commonly used for models where one has to predict the probability of an output. Since the probability of something exists only between 0 and 1, the sigmoid is the correct selection due to its range;

- As shown in Figure 1.7, the function is differentiable and provides a smooth gradient. This is expressed as an S-shape of the sigmoid activation function

The limitations of the sigmoid function are:

- its derivative, since the gradients values are only significant for the range $[-3,3]$ It implies that outside of this region the gradient is very small and the network stops learning (vanishing gradient problem);

- the asymmetry of the output close to zero. The output of the neurons will be of the same sign. This makes the training of the neural network more difficult.

ReLU function - Rectified Linear Unit

The ReLU function looks like a linear function and admits a derivative function. This allows for backpropagation and makes it computationally efficient.

The catch is that the ReLU function doesn't activate all neurons simultaneously. The neurons will be deactivated only if the output is less than zero. Figure 1.8 shows the ReLU function and mathematically it can be represented as:

$$f(x) = \max(0, x) \tag{1.6}$$

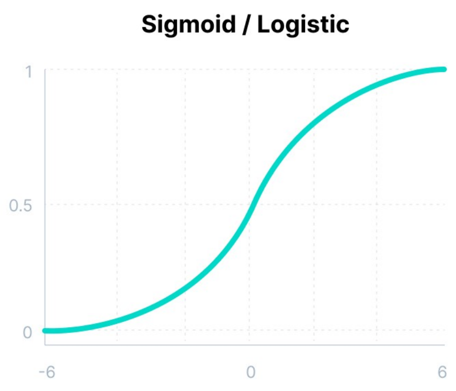


Figure 1.6: Sigmoid function [9].

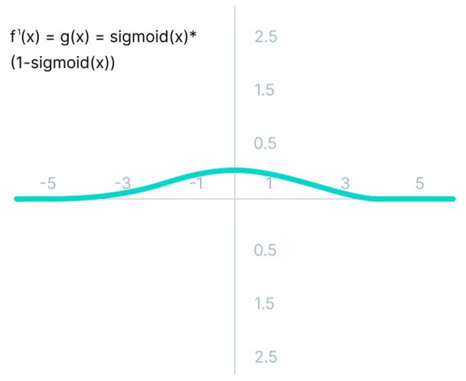


Figure 1.7: Derivative of the sigmoid function [9].

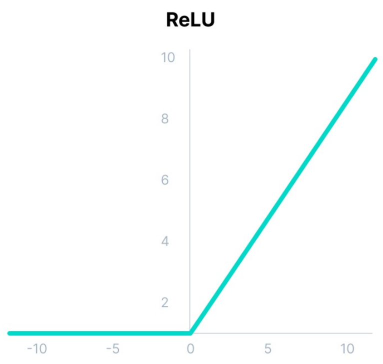


Figure 1.8: Representation of the ReLU function [9].

The advantages of using a ReLU as an activation function are:

- A greater efficiency since only a certain number of neurons are activated by this function;

- Due to its linearity and non-saturation property, ReLU accelerates the convergence of gradient descent to the global minimum of the loss function.

The big problem with ReLU is that its derivative before zero is zero. This means that it has a zero gradient. Because of that, the weights and biases for some neurons are not updated during the backpropagation process.

To face this problem other activation functions exist such as the leaky ReLU, the parametric ReLU, and exponential ReLU (ELUs) [9].

Gradient Descent

The gradient descent algorithm is used in supervised learning and involves adjusting the network weights and biases by propagating back the error from the output to the input layer. The basic idea behind the gradient descent algorithm is to minimize the difference between the current solution and the actual network solution. This is enabled by changing the network weights through an optimization algorithm.

Two main phases can be observed in the optimization algorithm. The forward pass is the first phase, the network is fed with input data, and the current output for these inputs is calculated with the current weights. The second phase, on the other hand, is based on the comparison of the current output (computed in the forward pass) and the output that would have been expected (which can come from a dataset, for example). The error between the two solutions is first calculated in the output layer and secondly propagated backward towards the other layers. Based on the error, each layer's weights are updated in order to minimize it. This process is repeated many times until the error is minimized.

The method used to compute the gradient backward is named backpropagation and is a part of the gradient descent. The cost function is defined as the difference between the two outputs. The gradient computed is the cost function gradient with respect to the inputs.

Gradient descent is a powerful technique but it can suffer of slowness and very high computational costs. It can also experience overfitting if not properly regularized.

Learning Rate

One of the most important hyperparameters that should be set is the learning rate. Hyperparameters are set before the training starts and are able to control the behavior and performance of the network. Some hyperparameters are the number of hidden layers, the number of neurons in each hidden layer, the batch size, and the learning rate.

The learning rate scales the magnitude of the weights update which aims to minimize the loss function. If the learning rate is too low the training process will take a long time and if is too high can lead to non-convergence in the loss function. In Figure 1.9 it is shown what was previously said. In Equation 1.7 is shown the weights update with the learning rate (η).

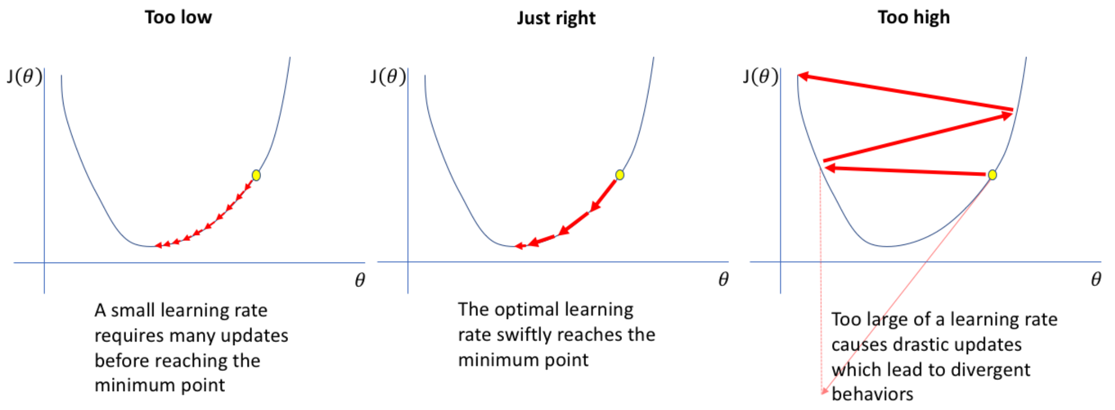

Too low
Just right
Too high
J(θ)
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Figure 1.9: Learning rate effects for the convergence [11].

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \eta \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) \tag{1.7}$$

where η is the learning rate, $\boldsymbol{w}_{i+1}$ are the updated weights, $\boldsymbol{w}_i$ are the previous weights and $\nabla_{\boldsymbol{w}} J(\boldsymbol{w})$ is the gradient of the loss function. This process can add also some momentum to speed up and decrease the convergence but since the discussion is very long please refer to reading in the literature.

1.4.2 Classification: support vector machines and random forests

Classification is a fundamental task in machine learning, which allows obtaining the labels or categories of a set of measurements from an *a priori* labeled dataset. To solve this problem two popular algorithms are Support Vector Machines (SVMs) and Random Forests. According to [2], the problem can be specified by the following loss function for both:

$$L\left[\vec{y}, \Phi(\vec{x}, \vec{y}, \vec{w})\right] = \begin{cases} 0, & \text{if } \vec{y} = \phi(\vec{x}, \vec{y}, \vec{w}) \\ 1, & \text{if } \vec{y} \neq \phi(\vec{x}, \vec{y}, \vec{w}) \end{cases} \tag{1.8}$$

Support vector machines is a powerful learning algorithm that can be used for both regression and classification. SVMs work by searching for the hyperplane that best separates the data into different classes. The plane is chosen to maximize the margin between these classes. SVMs are good for high-dimensional data and can deal with both linear and nonlinear separations using kernel functions. Typically SVMs are used for image and text classification.

Random forests, on the other hand, are ensemble learning methods that combine multiple decision trees in order to improve accuracy. Each tree is constructed using a subset of features and a random subset of training data. The output will be given by the most common vote among all the decision trees. In other words, each decision tree predicts a class and vote, and the class with the most votes is the most likely. Random forests are effective in dealing with noisy data and are less prone to overfitting than a single decision tree.

1.5 Unsupervised Learning

Unsupervised learning is a type of machine learning in which the algorithm is not provided with labeled data. In contrast to supervised learning, in which the algorithm was given labeled data in this case it is the algorithm that has to find patterns and structures in the input dataset provided without an *a priori* knowledge of what the output should look like. Unsupervised learning aims to extract and identify meaningful patterns in the input dataset that can be used to extract information, such as hidden relationships between variables or clusters of similar data points. This technique, therefore, is principally used for dimensional reduction, clustering, and anomaly detection [2].

1.5.1 Dimensional reduction: proper orthogonal decomposition, principal component analysis, and auto-encoders

The extraction of flow features from experimental data and large-scale simulations is very important in the fluid dynamic framework. Even, the identification of low-dimensional structures for high-dimensional data can be used as preprocessing for all other tasks in the supervised learning context.

Proper Orthogonal Decomposition (POD) is a mathematical technique that aims to decompose the dataset into a base of orthogonal vectors. This technique is used to represent the data in a smaller dimensional space while preserving as much information as possible. This technique can be used to analyze data with a spatial and temporal structure.

Principal Component Analysis (PCA) is another popular technique for dimensionality reduction. This technique works to find the principal components of a dataset, which are the directions in which the data varies the most. These components can be used by projecting the dataset data into this new lower-order dimensional space preserving as much variance as possible. The PCA algorithm will be further explained in subsection 3.4.2.

Auto-encoders are a particular type of neural network (they can be seen as a two-layer neural network). They consist in an encoder which maps the input data to a low-order dimensional space and a decoder, which maps the encoded data back to the original space.

While each of these techniques has its strengths and weaknesses, they share the goal of reducing the complexity of a dataset while preserving its most important information. In recent years, the machine learning community has developed a wide number of autoencoders that, when properly matched with the possible features of the flow field can lead to significant insight for reduced-order modeling of stationary and time-dependent data [2].

1.5.2 Clustering and vector quantization

Clustering is a common technique in unsupervised machine learning which aims to identify similar groups in the data [2]. The most widely used algorithm is the k-means clustering which partitions data into k clusters.

Vector quantizers identify representative points for data that can be partitioned into a pre-determined number of clusters. In the latter, this point can be used instead of the full dataset for future samplings. The vector quantizer $\Phi(\vec{x}, \vec{w})$ provides a mapping from the data $\vec{x}$ and the coordinate of the clusters. The loss function, according to [2], in this case, can be written in order to minimize the distance from the data to the cluster center, as follows:

$$L\left[\Phi(\vec{x}, \vec{y}, \vec{w})\right] = ||x - \Phi(\vec{x}, \vec{w})||^2$$

Clustering and vector quantization are powerful techniques that can be used to gain insights and make predictions from large datasets. By grouping similar data points into clusters, these techniques can reveal hidden patterns and structures in the data that may not be apparent from the raw data alone.

1.6 Semisupervised Learning

Semisupervised learning operates under partial supervision, it works with both labeled and unlabeled data. In semisupervised learning, the model is trained with mixed input data whose goal is to learn a general representation of the data. This representation is used to make predictions about new ones.

A common approach is to use a combination of supervised and unsupervised learning techniques. For example, a popular approach is to use a deep neural network (seen in subsection 1.4.1) that is trained with a combination of labeled and unlabeled data. The supervised objective is used to maximize the model's predictions on labeled data, and the unsupervised objective is used to help the model to learn a good representation of the data distribution.

In the fluid dynamics context, two algorithms can be found: generative adversarial networks (GANs) and Reinforcement Learning (RL).

1.6.1 Generative Adversarial Networks (GANs)

This type of network was first introduced by Goodfellow in [12]. The basic idea of this method is to train two neural networks that compete with each other simultaneously: a generator and a discriminator. The generator takes random inputs and produces a sample with the respective inputs. The discriminator is acting as a classifier that evaluates the generator's work. The goal is for the generator to find a way to create more realistic samples and for the discriminator to try to improve its ability in classifying what is true and what is a synthetic sample. The optimal point is achieved when the synthetic samples produced by the generator are very similar to the real ones and the discriminator can't work anymore. The weights are obtained in a process which is inspired by game theory, called adversarial learning. The discriminator is first fed with trained data, this is why this procedure is often called self-supervised [2].

GANs can be used to generate realistic simulations of turbulent flows [13] or to control and optimize a fluid system.

1.6.2 Reinforcement Learning (RL)

Reinforcement Learning (RL) is a mathematical framework for problem-solving and focuses on the development of algorithms that can learn how to make decisions in a dynamic environment. It is composed of an agent who is not provided with labeled information but learns to interact with the environment in the form of rewards or penalties. The agent's goal is to learn a policy that maximizes the reward.
The key components of the RL systems are:

- The State which is the state of the environment that is used by the agent to decide what action should be taken;
- The Action which is the decision made by the agent with the given state. After the action the state will be changed;
- The Reward which is the feedback given to the agent if an efficacious action is carried out;
- The Policy which is the final strategy learned. From any given state the policy decides what action should be taken. The goal is to learn this policy for the agent.

Reinforcement learning was applied in several fluid dynamics problems such as it was used to learn a policy to reduce the drag in a turbulent boundary layer in [14].

1.7 Turbulent Flows Introduction

Most flows encountered in engineering practice are turbulent [15]. Turbulent flows are characterized by the following properties:

- Turbulent flows are unsteady. In a plot, for example, of the velocity profile as a function of the time can be noted clearly the chaotic nature of the turbulence [16];
- Is a three-dimensional flow. The instantaneous field fluctuates into 3 dimensions although the time average can be done in only two dimensions. For a strictly two-dimensional flow, vortex stretching cannot be done and consequentially energy cascade to the smaller scales cannot take place [15];
- Turbulent flows contain a great deal of vorticity. Indeed vortex stretching mechanism is the main actor to increase the intensity of the turbulence;
- Turbulent flows dissipates energy. The dissipation zone is located mostly on the fine scales instead of on the larger scales where there is the majority of turbulent kinetic energy;
- Turbulent flows fluctuate on a broad range of lengths and time scales. This property makes direct numerical simulation very difficult;
- Turbulence increases the rate at which conserved quantities are stirred. The eddy motion produces the transport of momentum and energy much higher than the molecular motion. The actual mixing is accomplished by diffusion (turbulence diffusion).

In the past, the primary approach to studying turbulence flows was experimental. Overall parameters such as the time-averaged drag or the heat transfer are relatively easy to measure. Other types of measurements are impossible to make such as measuring the fluctuating pressure. Numerical methods can be helpful to get this type of information. There are some approaches to predict turbulent flows:

- The first one is based on the integral equations which can be derived from the equation of motion by integrating over one or more coordinates;
- The second approach is based on decomposing the equation of motion into a mean and fluctuating part. Unfortunately, these decomposed equations do not form a closed set of equations and these methods require the introduction of approximation (*Turbulence models*). Some turbulence models can be observed in chapter 2. The actual approach to handling the turbulence models is dictated by the nature of the process used to obtain the mean and the fluctuating

equations. A set of partial differential equations called Reynolds - Averaged Navier Stokes (RANS) can be obtained if the process to create the mean is averaging (over time or an ensemble of realizations) or a set of equations called Large Eddy Simulation LES can be obtained when the mean is achieved averaging or filtering over finite volumes in space. An accurate description of these types of simulations was taken into account in section 2.1

- The last approach is the Direct Numerical Simulations (DNS) in which the Navier Stokes equations are solved for all the motions in turbulent flows. This approach is described below.

Direct Numerical Simulations (DNS)

This method consists in solving the Navier Stokes equations without averaging or approximation. It is also the simplest approach from the conceptual point of view.

In a direct numerical simulation, to assure all of the structures of the turbulence have been captured, the domain considered should be at least large as the largest eddy [16]. A useful measure for the larger scale is the integral scale (L) which is the distance over which the fluctuating component of the velocity remains correlated [16].

According to [15], denoting L_{11} as the longitudinal integral length scale, for an isotropic turbulence with a given spectrum, the reasonable lower bound on the box size is $\mathcal{L} = 8L_{11}$. In term of the lowest wavenumber k_0 implies that [15]:

$$k_0 L_{11} = \frac{2\pi}{\mathcal{L}} \frac{\mathcal{L}}{8} \approx 0.8 \tag{1.9}$$

Since it is necessary to capture all of the kinetic energy dissipation that occurs at the smallest scale a good simulation must have a fine enough grid. This scale is called the Kolmogorov scale (η). Usually, the resolution required for the smallest scales or the maximum wavelength is:

$$k_{\max}\eta = \frac{\pi}{\Delta x}\eta \geq 1.5 \tag{1.10}$$

where $k_{\max}$ is the largest wavenumber, Δx is the grid size and η the dimension of the Kolmogoroff scale.

The time step in a DNS simulation is also important, the particles must move only in a fraction of the grid spacing (Δx) in the time step (Δt). The Courant number, in according to [15], is imposed:

$$\text{CFL} = \frac{k^{1/2}\Delta t}{\Delta x} \tag{1.11}$$

The computational cost increases as Re_L^3 (Reynolds number based on the magnitude of the velocity fluctuation and the integral scale) and about Re_λ (Reynolds number based on the Taylor scale).

Chapter 2

Machine Learning at Low Prandtl Numbers

In the research group in which this thesis work was developed, the objective was to characterize the thermal-hydraulics of a liquid metal-cooled nuclear reactor. Dealing with this topic, the study of heat transfer turns out to be very interesting since the nominal operating conditions induce the Prandtl number of the liquid metals (lead or sodium) to be very low. In this system, the thermal-hydraulics in normal and transient conditions must be carefully analyzed first for the system's safety and then for the project design.

In this respect, Computational Fluid Dynamics (CFD) is regarded as a valuable tool to tackle the challenges associated with the thermal-hydraulic behavior of nuclear systems [17]. In the Reynolds-Averaged Navier-Stokes (RANS) framework, a large number of models for the closure of the turbulent momentum flux exist. On the other hand, limited availability of RANS thermal models are present in commercial codes since a common use is to introduce a constant turbulent Prandtl number (Pr_t) by applying the so-called Reynolds analogy to obtain the thermal diffusivity.

In the presence of forced convection of common fluids, such as water or air, which have Prandtl numbers close to or greater than unity, this approach is widely accepted. However, in the case of liquid metals, which are characterized by a low Prandtl number, the hypothesis of similarity between the momentum and the thermal turbulence is less justified.

This chapter will first show how to solve this problem with modern CFD techniques suitable for low Prandtl fluids and then how it could be solved using a machine learning technique.

2.1 CFD techniques at Low Prandtl numbers

CFD methods can be grouped into three categories: Direct Numerical Simulation (DNS), Large Eddy Simulation (LES), and Reynolds-Averaged Navier–Stokes simulation (RANS). While DNS and LES offer high calculation accuracy, they require significant computational resources and are only suitable for certain, simple geometric models.

Since the computational cost of the RANS approach is much lower than that of DNS and LES, the RANS approach is the most widely used CFD method in engineering calculations. The Reynolds decomposition to the velocity and the temperature can be applied in this way :

$$\begin{cases} \hat{U} = U + u' \\ \hat{\theta} = \theta + \theta' \end{cases} \tag{2.1}$$

where U and θ represent respectively the mean velocity and the mean temperature and u' and θ' represent respectively the fluctuating velocity and temperature.

For incompressible turbulent flows with constant physical properties and no gravity the Reynolds averaged equations are shown in Equation 2.2:

$$\begin{cases} \dfrac{\partial U_i}{\partial x_i} = 0 \\ \dfrac{\partial U_i}{\partial t} + U_j \dfrac{\partial U_i}{\partial x_j} = -\dfrac{1}{\rho}\dfrac{\partial P}{\partial x_i} + \dfrac{\partial}{\partial x_j}\left(\nu \dfrac{\partial U}{\partial x_j}\right) - \dfrac{\partial \overline{u_i' u_j'}}{\partial x_j} \\ \dfrac{D\theta}{Dt} = \alpha \dfrac{\partial^2 \theta}{\partial x_j \partial x_j} - \dfrac{\partial \overline{u_i' \theta'}}{\partial x_j} \end{cases} \tag{2.2}$$

where ν, U_i, P, θ_i and α are respectively the molecular viscosity, the Reynolds-averaged velocity, the pressure, the Reynolds-averaged temperature, and the molecular diffusivity. $\overline{u_i' u_j'}$ is the so-called Reynolds stress tensor and $\overline{u_i' \theta'}$ is the turbulent heat flux.

This problem is not closed since more unknowns than equations are present, it is necessary to introduce a closure for the momentum field and one for the thermal field.

In the RANS framework, to accurately predict the momentum transport of various fluids, the linear eddy viscosity $k - \epsilon$ or $k - \omega$ turbulence model is usually sufficient [18]. On the other hand, the Reynolds analogy hypothesis (constant turbulent Prandtl number $Pr_t \approx 0.85 - 0.9$) is used to reproduce the heat transfer in almost all commercial codes [19]. For the simulation of fluids like water and hair (with a relatively high Pr number), rational results can be obtained with a constant Pr_t. However with high thermal diffusivity and low viscosity, resulting in a low Pr number ($Pr \approx 0.01 - 0.025$), the Reynolds analogy hypothesis is no longer appropriate (Figure 5.8).

Subsequently, this section will discuss the most frequently used methods for closing the thermal and moment part.

2.1.1 Turbulence Model for Momentum Field

For the momentum part, the Reynolds stresses are modelled in the terms of the eddy viscosity as follows:

$$-\rho\overline{u_i'u_j'} = \mu_t\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right) + \frac{2}{3}\rho k\delta_{ij} \tag{2.3}$$

where μ_t is the eddy viscosity, k is the turbulent kinetic energy and δ_{ij} the Kronecker delta.

As it can be seen in Equation 2.3 the unknown was shifted from the Reynolds stresses tensor to the eddy viscosity. The eddy viscosity is defined as a function of the turbulent kinetic energy k, and the turbulent dissipation rate ϵ as :

$$\mu_t = c_\mu f_\mu \rho\frac{k^2}{\epsilon} \tag{2.4}$$

where c_μ is a constant determined in the equilibrium analysis at high Reynolds numbers and the dumping function f_μ will be a function of turbulence Reynolds numbers $Re_t = \rho k^2/\epsilon\mu$.

Other two equations should be added: one for turbulent kinetic energy k and one for the turbulent dissipation ϵ as follows [20]:

$$\begin{cases} \dfrac{\partial\rho k}{\partial t} + \dfrac{\partial}{\partial x_j}\left[\rho u_j\dfrac{\partial k}{\partial x_j} - \left(\mu + \dfrac{\mu_\tau}{\sigma_k}\right)\dfrac{\partial k}{\partial x_j}\right] = \tau_{ij}^{\text{turb}}S_{ij} - \rho\epsilon + \phi_k \\ \dfrac{\partial\rho\epsilon}{\partial t} + \dfrac{\partial}{\partial x_j}\left[\rho u_j\epsilon - \left(\mu + \dfrac{\mu_\tau}{\sigma_\epsilon}\right)\dfrac{\partial\epsilon}{\partial x_j}\right] = c_{\epsilon1}\dfrac{\epsilon}{k}\tau_{ij}^{\text{turb}}S_{ij} - c_{\epsilon2}f_2\rho\dfrac{\epsilon^2}{k} + \phi_\epsilon \end{cases} \tag{2.5}$$

where $c_{\epsilon1}$, $c_{\epsilon2}$, c_μ , σ_k and σ_ϵ are the model constants, f_μ, f_1 and f_2 are the damping functions and $S_{ij} = \frac{1}{2}\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right)$ is the rate of strain tensor. All of these constants and functions are defined as [20]:

- $c_{\epsilon1} = 1.45$;
- $c_{\epsilon2} = 1.92$;
- $c_\mu = 0.09$;
- $\sigma_k = 1$;

- $\sigma_\epsilon = 1.3$;
- $f_\mu = \exp\left(-3.4/(1 + 0.02Re_t)^2\right)$;
- $f_2 = 1 - 0.3\exp\left(-Re_t^2\right)$;
- $\phi_k = 2\mu\left(\frac{\partial\sqrt{k}}{\partial y}\right)^2$;
- $\phi_\epsilon = 2\mu\frac{\mu_t}{\rho}\left(\frac{\partial^2 u_s}{\partial y^2}\right)^2$

where u_s is the flow velocity parallel to the wall. At this point, the problem is closed since a sufficient number of equations are present.

2.1.2 Turbulence Models for Thermal Field

For the thermal part, the turbulent heat flux $\overline{u_i'\theta'}$ can be modeled in many ways. The most common one, in the case of a unitary Pr number, is to apply the Reynolds analogy which cannot be applied in the case of low Prandtl number. Other closure models were applied such as the Manservisi and Menghini model [19].

Both the Reynolds analogy and the Manservisi model start from the Simple Gradient Diffusion hypothesis (SGD) shown in Equation 2.6:

$$\overline{u_i'\theta'} = -\alpha_t\frac{\partial\theta}{\partial x_i} \tag{2.6}$$

where α_t is the turbulent diffusivity that should be modeled.

Reynolds Analogy closure model

The Reynolds analogy assumption is made to model the turbulent diffusivity α_t and allows the closure for the turbulent heat flux.

$$\alpha_t = -\frac{\nu_t}{Pr_t} \tag{2.7}$$

where $\nu_t = \frac{\mu_t}{\rho}$ can be obtained from Equation 2.4 and the turbulent Prandtl number is supposed to be a constant value i.e. $Pr_t \approx 0.85$. The Equation 2.6 is closed and the problem can be solved.

Manservisi and Menghini closure model

The Manservisi and Menghini closure model is a two-equation (k_θ and ϵ_θ) problem. This closure model is vastly used in the thermohydraulic of lead-cooled fast reactors which include the average square temperature fluctuation k_θ and its dissipation ϵ_θ.

In the Manservisi $k_\theta - \epsilon_\theta$ model the SGD hypothesis was applied and α_t is computed as follows [18]:

$$\alpha_t = C_\theta k \tau_{l\theta} \tag{2.8}$$

where C_θ is an empirical constant, τ_θ is the local thermal characteristic time modelled as follows [18]:

$$\tau_{l\theta} = f_{1\theta} B_{1\theta} + f_{2\theta} B_{2\theta} \tag{2.9}$$

in which:

$$\begin{cases} f_{1\theta} = \left[1 - \exp\left(-\dfrac{R_\delta}{19\sqrt{Pr}}\right)\right]\left(1 - \exp\left(-\dfrac{R_\delta}{14}\right)\right) \\ B_{1\theta} = 0.9\tau_{\mathrm{u}} = 0.9\dfrac{k}{\varepsilon} \\ f_{2\theta}B_{2\theta} = \tau_u\left(f_{2a\theta}\dfrac{2R}{R + C_\gamma} + f_{2b\theta}\sqrt{\dfrac{2R}{Pr}}\dfrac{1.3}{\sqrt{\mathrm{Pr}}R_t^{3/4}}\right) \\ f_{2a\theta} = f_{1\theta}\exp\left[-\left(\dfrac{R_t}{500}\right)^2\right] \\ f_{2b\theta} = f_{1\theta}\exp\left[-\left(\dfrac{R_\delta}{200}\right)^2\right] \end{cases} \tag{2.10}$$

where $R_\delta = \delta\ {\epsilon^{0.25}}/{\nu^{0.75}}$, $\tau_u = {k}/{\epsilon}$, $R_t = {k^2}/{\nu\epsilon}$ and $R = {\tau_\theta}/{\tau_u}$. The equation for k_θ and ϵ_θ can be written as follows [18]:

$$\begin{cases} \dfrac{\partial k_\theta}{\partial t} + u_j\dfrac{\partial k_\theta}{\partial x_j} = \dfrac{\partial}{\partial x_j}\left(\left(\alpha + \dfrac{\alpha_t}{\sigma_{k_\theta}}\right)\dfrac{\partial k_\theta}{\partial x_j}\right) + \alpha_t\left(\dfrac{\partial T}{\partial x_j}\right)\left(\dfrac{\partial T}{\partial x_j}\right) - \varepsilon_\theta \\ \dfrac{\partial \varepsilon_\theta}{\partial t} + u_j\dfrac{\partial \varepsilon_\theta}{\partial x_j} = \dfrac{\partial}{\partial x_j}\left(\left(\alpha + \dfrac{\alpha_t}{\sigma_{\varepsilon_\theta}}\right)\dfrac{\partial \varepsilon_\theta}{\partial x_j}\right) + \dfrac{\varepsilon_\theta}{k_\theta}(C_{P1}P_\theta - C_{d1}\varepsilon_\theta) + \\ \qquad + \dfrac{\varepsilon_\theta}{k}(C_{P2}P_k - C_{d2}\varepsilon) \end{cases} \tag{2.11}$$

where:

$$C_{d2} = \left\{1.9\left[1 - 0.3\exp\left(-\left(\frac{R_t}{6.5}\right)^2\right)\right] - 1\right\}\left[1 - \exp\left(-\frac{R_\delta}{5.7}\right)\right]^2 \tag{2.12}$$

All the constants applied in the above equations can be found in Table 2.1.

C_θ	C_γ	$\sigma_{k\theta}$	$\sigma_{\epsilon\theta}$	C_{p1}	C_{d1}	C_{pw}
0.1	0.3	1.4	1.4	0.925	1	0.9

Table 2.1: Manservisi and Menghini closure coefficients

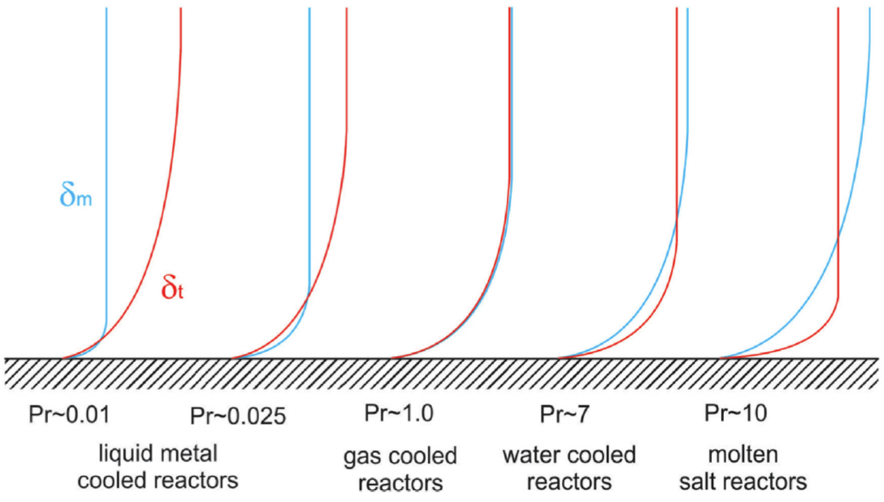


Figure 2.1: Comparison of momentum δ_m and thermal δ_t boundary layer in different working fluids [21].

It is shown in Figure 2.1 that applying the Reynolds analogy in a liquid metal cooled nuclear reactor is a very strong assumption since there is no equality in the momentum and the thermal boundary layer [21].

Models such as Manservisi and Menghini, shown above, were created to overcome this problem. Although these models can produce outstanding results, they are challenging to implement, so a machine learning model was sought in this thesis to overcome this problem.

2.2 Machine learning techniques at Low Prandtl numbers

This section presents an alternative way to solve the problem using an algebraic heat flux model obtained from a data-driven artificial neural network. This is because current CFD techniques cannot be applied at low Prandtl numbers (Reynolds analogy) or are very difficult to implement (Manservisi and Menghini).

Recent developments in machine learning have brought excellent tools for summarising and translating DNS data into mathematical forms. However, the accuracy of these models depends very much on the way the data is selected and processed, the algorithm, and the training procedures [1].

Artificial Neural Networks (ANNs) have become a popular technique for solving high-dimensional and highly non-linear regression problems in turbulence modelling. However, the ANN approach presents challenges related to generalization and practical applicability and, if not properly trained and regularised, can lead to non-physical results. To increase the robustness of the model, essential physical and mathematical properties should be incorporated indirectly or by construction, such as the invariance under rotation, the realizability of the Reynolds stresses, and the consistency with thermodynamics [1].

The formulation of the inputs and the structure of the neural network was previously carried out by M. Fiore in the paper [1] and I will limit myself only to a description of the past work as an introduction of work described in the next chapters.
In this section the formulation and the tuning of the network will be covered.

2.2.1 Mathematical formulation

It is required a model for the turbulent heat flux $\overline{u_i'\theta'}$ which is a three components vector. The averaged Reynolds equation for incompressible flows with constant properties for the temperature is the following:

$$\frac{D\theta}{Dt} = \alpha \frac{\partial^2 \theta}{\partial x_j \partial x_j} - \frac{\partial \overline{u_i'\theta'}}{\partial x_j} \tag{2.13}$$

which is the same as Equation 2.2 remembering the notation for the velocity and the temperature in Equation 2.1. The regression problem consists in finding the value of the neural network structure (weights and biases) that minimizes the value of the turbulent heat flux in comparison with the DNS value.

The algebraic equation for the turbulent heat flux was derived as a function of scalars, vectors, and tensors quantities in this way [1]:

$$\overline{\boldsymbol{u'\theta'}} = f\left(\boldsymbol{b}, \boldsymbol{S}, \boldsymbol{\Omega}, \nabla\theta, \mathbf{g}, k, \epsilon, k_\theta, \epsilon_\theta, \alpha, \nu\right) \tag{2.14}$$

where the tensors $\mathbf{b}$, $\mathbf{S}$ and $\mathbf{\Omega}$ are defined in Equation 2.15, $\mathbf{g}$ is the gravity vector, k is the turbulent kinetic energy, ϵ is the turbulent dissipation rate, k_θ is the thermal variance, ϵ_θ is the thermal dissipation rate, α the molecular diffusivity and ν the molecular viscosity.

$$\begin{aligned} b_{ij} &= \frac{\overline{u'_i u'_j}}{k} - \frac{2}{3}\delta_{ij} \\ S_{ij} &= \frac{1}{2}\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right) \\ \Omega_{ij} &= \frac{1}{2}\left(\frac{\partial U_i}{\partial x_j} - \frac{\partial U_j}{\partial x_i}\right) \end{aligned} \tag{2.15}$$

Equation 2.14 implies :

- Locality and instantaneously as a consequence of the algebraic structure. It can also be convenient since it avoids a second-order closure for the turbulent heat flux [1];
- Turbulence equilibrium since is assumed that no gradients of k, k_θ and b_{ij} are involved [1].

In addition, the dependence on gravity can be removed since the study can be limited to forced convection flows. The model being algebraic and explicit implies that can be written as the product of a dispersion tensor $\boldsymbol{D}$ and the mean temperature gradient ($\nabla\theta$):

$$\overline{\boldsymbol{u'\theta'}} = -\boldsymbol{D}\nabla\theta \tag{2.16}$$

where from Equation 2.14 the dependency can be shifted into $\boldsymbol{D}$ assuming also that the anisotropic part of $\boldsymbol{D}$ is a function only of the anisotropic part of the momentum field [1]. $\boldsymbol{D}$ can be written as follows:

$$\boldsymbol{D} = \mathcal{F}\left(\boldsymbol{b}, \boldsymbol{S}, \boldsymbol{\Omega}, ||\nabla\theta||, k, \epsilon, k_\theta, \epsilon_\theta, \alpha, \nu\right) \tag{2.17}$$

where $||\cdot||$ is the norm.

Since the quantity to be modelled is physical the model should respect the invariance under the rotation property of the coordinate system. This property can be applied in the following way:

$$\boldsymbol{QDQ}^T = \mathcal{F}\left(\boldsymbol{QbQ}^T, \boldsymbol{QSQ}^T, \boldsymbol{Q\Omega Q}^T, ||\nabla\theta||, k, \epsilon, k_\theta, \epsilon_\theta, \alpha, \nu\right)$$

where $\boldsymbol{Q}$ is any rotational matrix.

Considering a new set of general tensors $\boldsymbol{T}^i$ that are formed through the product of the tensors $\boldsymbol{b}$, $\boldsymbol{S}$ and $\boldsymbol{\Omega}$ a linear mapping between $\boldsymbol{D}$ and $\boldsymbol{T}$ can be made as shown in Equation 2.18 where the coefficients c_i are isotropic functions.

$$\boldsymbol{QDQ}^T = \sum_{i=1}^{n} c_i \boldsymbol{QT}^i\boldsymbol{Q}^T \tag{2.18}$$

An infinite number of tensors product can be formed with $\boldsymbol{b}$, $\boldsymbol{S}$ and $\boldsymbol{\Omega}$ leading to an infinite expansion series. However, the tensor representation theory [22] demonstrated that exists a finite set of independent invariants under the proper orthogonal group for a finite tensorial set.

Can be shown in Fiore et al. [1] that the minimal tensors basis is formed by the following set:

$$\boldsymbol{I}, \boldsymbol{b}, \boldsymbol{S}, \boldsymbol{\Omega}, \boldsymbol{bS}, \boldsymbol{S\Omega}, \boldsymbol{bS\Omega}$$

and the corresponding minimal basis of invariants is the following:

$$\{\boldsymbol{b_2}\}, \{\boldsymbol{b^2}\}, \{\boldsymbol{S^2}\}, \{\boldsymbol{\Omega^2}\}, \{\boldsymbol{bS}\}, \{\boldsymbol{bS\Omega}\}$$

where the notation $\{\cdot\}$ indicates the trace of the tensor and $\boldsymbol{b_2}$ is defined in Equation 2.19 where b_{ij} was defined in Equation 2.15 .

$$\mathbf{b} = \begin{pmatrix} b_{11} & b_{12} & 0 \\ b_{12} & b_{22} & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.19}$$

From Equation 2.18 the coefficients c_i can be deducted as:

$$c_i = f_i\left(\{\boldsymbol{b_2}\}, \{\boldsymbol{b^2}\}, \{\boldsymbol{S^2}\}, \{\boldsymbol{\Omega^2}\}, \{\boldsymbol{bS}\}, \{\boldsymbol{bS\Omega}\}, ||\nabla\theta||, k, \epsilon, k_\theta, \epsilon_\theta, \alpha, \nu\right)$$

and applying the Buckingham theorem, 10-dimensional independent groups were formed, and then the coefficients c_i can be written as follows:

$$c_i = f_i\left(\pi_i, Re_t, Pr\right) \tag{2.20}$$

The complete formulation of each one of the c_i coefficients can be seen in Table 2.2.

It is of fundamental importance to observe the second thermodynamics principle, which requires the real part of the eigenvalues of $\boldsymbol{D}$ to be positive. In forced convection regimes, this characteristic averts counter-gradient heat fluxes. To satisfy this requirement a Cholesky decomposition of the symmetric part of $\boldsymbol{D}$ should be enforced. $\boldsymbol{D}$ can be written as:

$$\boldsymbol{D} = \left[\left(\boldsymbol{A} + \boldsymbol{A}^T\right)\left(\boldsymbol{A} + \boldsymbol{A}^T\right) + \frac{k}{\epsilon^{0.5}}\left(\boldsymbol{W} - \boldsymbol{W}^T\right)\right] \tag{2.21}$$

where $\boldsymbol{A}$ and $\boldsymbol{W}$ can be written as a functions of the tensors basis $\boldsymbol{T}^i$ and the coefficients a_i and w_i in following equations:

$$\begin{cases} \boldsymbol{A} = \sum_{i=1}^{n} a_i \boldsymbol{T}^i \\ \\ \boldsymbol{W} = \sum_{i=1}^{n} w_i \boldsymbol{T}^i \end{cases} \tag{2.22}$$

and the coefficients a_i and w_i can be derived from c_i.

Invariant Basis	**Tensor Basis**
$\pi_1 = \frac{k^2}{\epsilon^2}\{\boldsymbol{S}^2\}$, $\pi_2 = \frac{k^2}{\epsilon^2}\{\boldsymbol{\Omega}^2\}$, $\pi_3 = \frac{1}{\{\boldsymbol{b}^2\}}$,	$\boldsymbol{T_1} = \frac{k}{\epsilon^{0.5}}\boldsymbol{I}, \boldsymbol{T_2} = \frac{k}{\epsilon^{0.5}}\boldsymbol{b}, \boldsymbol{T_3} = \frac{k^2}{\epsilon^{1.5}}\boldsymbol{S}$
$\pi_4 = \frac{k}{\epsilon}\{\boldsymbol{bS}\}$, $\pi_5 = \{\boldsymbol{b_2}\}$,	$\boldsymbol{T_4} = \frac{k^2}{\epsilon^{1.5}}\boldsymbol{\Omega}$, $\boldsymbol{T_5} = \frac{k^2}{\epsilon^{1.5}}\boldsymbol{bS}$,
$\pi_6 = \frac{k^2}{\epsilon^2}\{\boldsymbol{bS\Omega}\}$, $\pi_7 = \frac{\|\|\nabla\theta\|\|}{\sqrt{k_\theta}}\frac{k^{1.5}}{\epsilon}$,	$\boldsymbol{T_6} = \frac{k^2}{\epsilon^{1.5}}\boldsymbol{b\Omega}$, $\boldsymbol{T_7} = \frac{k^3}{\epsilon^{2.5}}\boldsymbol{S\Omega}$,
$R = \frac{k_\theta \epsilon}{k\epsilon_\theta}$, $Re_t = \frac{k^2}{\epsilon\nu}$, $Pr = \frac{\nu}{\alpha}$,	$\boldsymbol{T_8} = \frac{k^2}{\epsilon^{2.5}}\boldsymbol{bS\Omega}$,

Table 2.2: Basis tensors and invariants formulation, computed according to [1]

2.2.2 Neural Network structure

Once the inputs had been derived, it is necessary to observe the artificial neural network structure. The coefficients c_i ($a_{1:8}$ and $w_{1:8}$) are modelled by the artificial neural network, which takes the input vector $\boldsymbol{X}$ constituted by the invariant basis shown in Table 2.2:

$$\boldsymbol{X} = [\pi_1, \ldots, R, Re_t, Pr] \tag{2.23}$$

and gives as output the coefficients a_i and w_i:

$$\boldsymbol{Y} = [a_1, \ldots, a_8, w_1, \ldots, w_8] \tag{2.24}$$

with a function ($\mathcal{M}$) that depends on the network's weights and the biases in this form:

$$\boldsymbol{Y} = \mathcal{M}(\boldsymbol{X}, \mathcal{W}) \tag{2.25}$$

where $\mathcal{W}$ is the array representing all the weights and biases of the network.

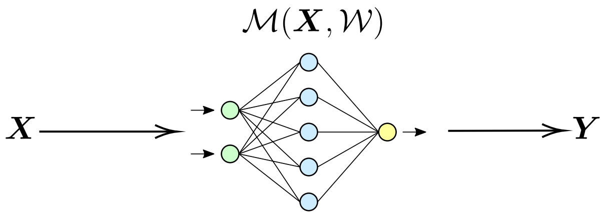


Figure 2.2: Simple representation of the ANN: take as inputs $\boldsymbol{X}$ and with the function $\mathcal{M}$ produce $\boldsymbol{Y}$.

Once the 16 coefficients a_i and w_i have been computed applying Equation 2.22 and Equation 2.21 the turbulent heat flux can be obtained.

The neural network structure, depicted in Figure 2.3, was created in the PyTorch framework and more in detail:

- Consists of two branches of which the first takes as inputs all the π_i groups, Re and R and the second one takes only the Pr number as input;
- All the inputs were logarithmically transformed;
- The first branch is constituted of six hidden layers with 100 neurons each which have a rectified linear unit as an activation function and an output layer with a hyperbolic tangent activation function;
- The second branch consists of two hidden layers with 100 neurons and each one with a ReLU function and the output layer has a hyperbolic tangent activation function;
- The output layer is a merge between the two output of the two branches by multiplication;
- The 16 coefficients a_i and w_i are computed and the turbulent heat flux can be obtained.

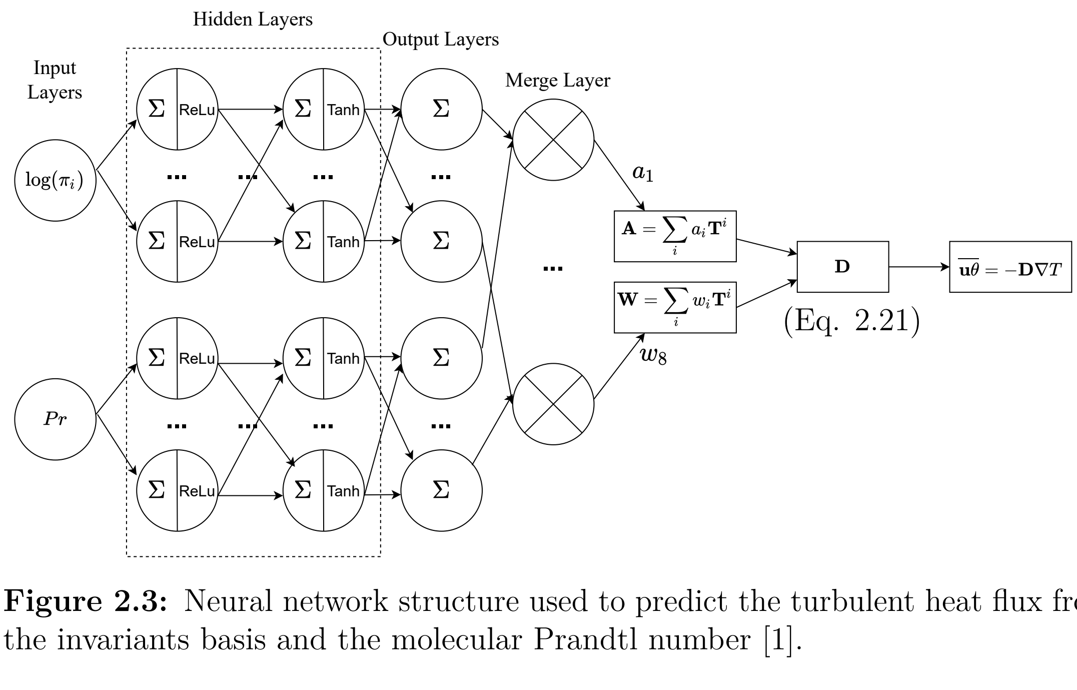

Hidden Layers
Output Layers
Input Layers
Merge Layer
$\log(\pi_i)$
Pr
Σ
ReLu
Tanh
a_1
w_8
$\mathbf{A} = \sum_i a_i \mathbf{T}^i$
$\mathbf{W} = \sum_i w_i \mathbf{T}^i$
D
(Eq. 2.21)
$\overline{\mathbf{u}\theta} = -\mathbf{D}\nabla T$

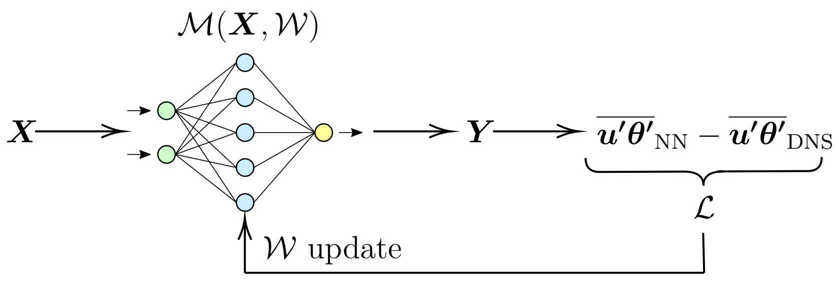
Figure 2.3: Neural network structure used to predict the turbulent heat flux from the invariants basis and the molecular Prandtl number [1].

The network was trained using the Adaptive Moment Estimator (ADAM) optimizer with a learning rate of 5×10^{-4} and a weight decay of 1×10^{-5}. The learning rate gives information about how much the weights are updated, a higher learning rate can speed up the convergence but can also escape from the minimum and diverge, and a lower learning rate increases the stability but decreases the convergence time (Figure 1.9). To address these issues, an inertial term, called momentum, that depends on the speed of descent was added to the optimizer. Weight decay is an L2 norm regularization that helps prevent overfitting. Specifically about overfitting, in the previous work [1] the network was tuned to avoid it. The network was trained with 25, 50, 100, and 200 neurons for each hidden layer and the results showed that the best choice is to have 100 neurons for each hidden layer to avoid both overfitting and underfitting.

2.2.3 Training of the neural network

The training phase of the neural network plays a crucial role. In this phase, the neural network learns how to make predictions based on input data. This process is made possible by updating the weights and biases, which then lead to different outputs. The weights and biases are constantly modified to reduce the error between the predicted output and the correct one (which in the present case is the DNS one).

In the present case, the training phase, as shown in Figure 2.4 consists, first, in a random initialization of the weights and the biases ($\mathcal{W}$), second, in evaluating the forward pass ($\boldsymbol{Y} = \mathcal{M}(\boldsymbol{X}, \mathcal{W})$) of the neural network, third, in computing the error between the predicted value of the neural network and the correct value coming from the DNS simulation, and, finally, in modifying the weights and the biases in order to minimize the previously computed error.



$\mathcal{M}(\boldsymbol{X}, \mathcal{W})$
$\boldsymbol{X}$
$\boldsymbol{Y}$
$\underbrace{\overline{\boldsymbol{u}'\boldsymbol{\theta}'}_{\mathrm{NN}} - \overline{\boldsymbol{u}'\boldsymbol{\theta}'}_{\mathrm{DNS}}}_{\mathcal{L}}$
$\mathcal{W}$ update

Figure 2.4: Training process of the artificial neural network.

It is of crucial importance to understand how to structure the loss function both in order to get around the minimum and also because one must guarantee the smoothness of the turbulent heat flux. The smoothness is required since the turbulent heat flux enters the averaged energy equation through its divergence as in Equation 2.13.

Based on what has been said, the loss function was obtained with the formulation shown in Equation 2.26 [1]:

$$\mathcal{L} = \frac{1}{N}\left(\sum_{i=1}^{N}\sum_{j=1}^{3}(\hat{q}_{i,j} - q_{i,j})^2\right) + \frac{\lambda}{N}\left(\sum_{i=1}^{N}\sum_{j,k=1}^{3}\left|\frac{\partial \hat{q}_{i,j}}{\partial x_k} - \frac{\partial q_{i,j}}{\partial x_k}\right|\Delta x_k\right) \tag{2.26}$$

where $i \in [1, \dots, N]$ is the index spanning across all the N data points contained in the mini-batch; $q_{i,j} = \overline{u'_j\theta'}$ is the turbulent heat flux prediction of the network, $\hat{q}_{i,j} = \overline{u'_j\theta'}_{\mathrm{DNS}}$ is the DNS turbulent heat flux and j is the index spanning in the 3 dimensions of the turbulent heat flux. In Equation 2.26 on the left is computed the difference between the true and the predicted heat flux and on the right the difference between the derivatives of those quantities with a regularizing parameter $\lambda = 10$ that is mesh independent.

2.2.4 State of the Art

The results obtained in Fiore et al. [1] are very accurate even when compared with different turbulent heat flux models. However, the trained neural network turns out to be very sensitive to the momentum turbulence model applied. Indeed, high accuracy is observed only with all momentum models that can accurately predict the Reynolds stresses i.e. with DNS data or the Elliptic-Blending Reynolds-Stress (EBRSM) model. On the contrary, if the neural network model is coupled with a momentum model that applies the Boussinesq hypothesis, inaccuracies are observed and can be seen in Table 2.3.

ANN Coupling State of the Art

Momentum Closure		**Thermal Closure**		**Accuracy**
DNS	+	Neural Network	=	Very High
EBRSM	+	Neural Network	=	Very High
$k - \epsilon$	+	Neural Network	=	Very Low

Table 2.3: State of the art of the coupling between momentum closure models and the artificial neural network trained in [1]. Good results had been achieved only if the artificial neural network was coupled with a very accurate momentum model.

Since the data-driven formulation does not seem to be robust enough against the deficiencies of the momentum modeling, a more robust model will be the goal of the present work. The new model must therefore be able to predict the correct value of the heat flux with good accuracy, even with inputs from a $k-\epsilon$ momentum turbulence model. The robustness, as can be observed in the following chapters, will be improved by perturbing the quality of the dataset by adding RANS data that apply the Boussinesq approximation.

Chapter 3

Input Computation and Analysis

In subsection 2.2.1 the correct formulation of the scalar inputs and for the tensor basis was demonstrated.

First, one must proceed in their calculation by also describing the database that will be used. Secondly, an analysis needs to be done to look at their shape and understand their location in space computing all the statistical values such as (mean, variance, skewness, and Kurtosis). A Principal Component Analysis (PCA) was also performed to see if it was possible to reduce the input space to use a reduced-order input dimension.

At the end of this chapter, a hyperparameter of the neural network will be tuned. This is the one responsible for the behavior of the network in presence of inputs coming from a DNS or RANS simulation

3.1 Database descriptions

As mentioned earlier the goal of this thesis is to obtain a closure of the thermal equation from inputs that are both DNS and especially RANS. In this situation, the database will consist of both DNS and RANS simulations related to different forced convection flows at various Reynolds and Prandtl numbers.

In the DNS case, the authors provided the momentum and the thermal statistics after time averaging the simulation results in a time interval related to the characteristic time of the flows.

In the RANS case, the simulations were performed by M.Fiore in the same conditions as the DNS case. Since we don't want to introduce errors in the RANS simulations due to a wrong thermal turbulence closure model, the temperature T, the thermal variance k_θ and the thermal dissipation rate ϵ_θ were taken from the DNS simulations in the same Reynolds and Prandtl conditions.

DNS Author and Ref.	**Flow configuration**	**Reynolds number**	**Prandtl number**
Kawamura et al. [23]	Channel Flow	$Re_\tau = 180$	Pr^1= 0.025 - 0.71
Kawamura et al. [23]	Channel Flow	$Re_\tau = 395$	Pr^2= 0.025 - 0.71
Kawamura et al. [23]	Channel Flow	$Re_\tau = 640$	Pr^2= 0.025 - 0.71
Tiselj et al. [24]	Channel Flow	$Re_\tau = 180$	$Pr = 0.01$
Tiselj et al. [24]	Channel Flow	$Re_\tau = 395$	$Pr = 0.01$
Tiselj et al. [24]	Channel Flow	$Re_\tau = 640$	$Pr = 0.01$
Oder et al. [25]	BFS	$Re_b = 3200$	$Pr = 0.01$

$Pr^1 = [0.025, 0.05, 0.1, 0.2, 0.4, 0.6, 0.71]$, $Pr^2 = [0.025, 0.71]$

Table 3.1: DNS databases for forced convection at different Re and Pr numbers that were used in the training and the validation of the neural network [23, 24, 25].

Channel flow test case

Kawamura and Tiselj provided two databases related to the isothermal channel flow configuration.

Kawamura et al. in [23] presented DNS results for a channel flow in these 4 configurations:

- Case 1 Poiseuille flow with constant wall heat flux heating,
- Case 2 Poiseuille flow with constant wall temperature difference,
- Case 3 Couette flow with constant wall heat flux heating,
- Case 4 Couette flow with constant wall temperature difference.

Only the Poiseuille flow with a constant wall heat flux for the Reynolds numbers and Prandtl numbers given in Table 3.1 was taken.

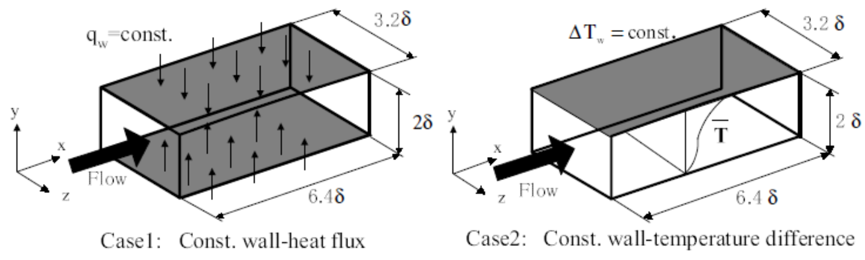


Figure 3.1: Computational domain and thermal boundary condition. [23]

The computational domain can be seen in Figure 3.1 and the flow was assumed to be fully developed. The coordinates and the flow variables were normalized by the channel half width δ, the friction velocity u_τ, the kinematic viscosity ν, and the friction temperature T_τ due to the constant heat flux condition [23].

The fundamental equations which are the continuity, the momentum, and the energy equation are shown as follows:

$$\begin{cases} \dfrac{\partial u_i^+}{\partial x_i^*} = 0, \\ \dfrac{\partial u_i^+}{\partial t^*} + u_j^* \dfrac{\partial u_i^+}{\partial x_j^*} = -\dfrac{\partial p^+}{\partial x_i^*} + \dfrac{1}{Re_\tau} \dfrac{\partial^2 u_i^+}{\partial x_j^{*2}} \\ \dfrac{\partial \theta^+}{\partial t^*} + u_j^+ \dfrac{\partial \theta^+}{\partial x_j^*} = \dfrac{1}{Re_\tau \cdot \mathrm{Pr}} \dfrac{\partial^2 \theta^+}{\partial x_j^{*2}} + u_1^+ \dfrac{\partial \left\langle \bar{T}_m^+ \right\rangle}{\partial x_1^*} \end{cases} \tag{3.1}$$

where the $(\ )^*$ superscripts means a normalization with δ, the $(\ )^+$ superscripts means a normalization with the friction velocity ν, $u_\tau = \sqrt{\tau_w/\rho}$ and $T_\tau = q_w/\rho c_p u_\tau$, the $\langle\ \rangle$ superscripts mean an averaging over the channel section, $Re_\tau = u_\tau \delta/\nu$ the Reynolds number, θ the transformed temperature and T_m the bulk mean temperature [23].

The inputs for the neural network were computed according to Table 2.2 and shown in Figure 3.2 and 3.3.

In Figure 3.2 and 3.3 can be distinguished: the DNS inputs computed, with the Kawamura database and, the RANS inputs, computed with the results of an OpeanFoam simulation with a $k-\epsilon$ momentum turbulence model and, the thermal part (T, ϵ_t, k_t), from the DNS corresponding simulation.

As can be seen in Figure 3.2 the inputs π_1 and π_2 are the same. This is due to the 1D hypothesis applied in the channel flow case which has been presented in section A.2. It can be noted also in Figure 3.3, that the inputs differ significantly between the DNS and RANS cases. This phenomenon is not due to an error in the calculation of the inputs, but to the fact that Boussinesq's hypothesis was applied in the RANS case since the $k-\epsilon$ turbulence model was applied, which leads to a totally different Reynolds stress tensor in the two cases. This behavior can be explained in section A.1 and will be the reason why a principal component analysis was done on the input space (subsection 3.4.3) and a feature importance analysis (section 6.3) was performed in both cases (DNS and RANS).

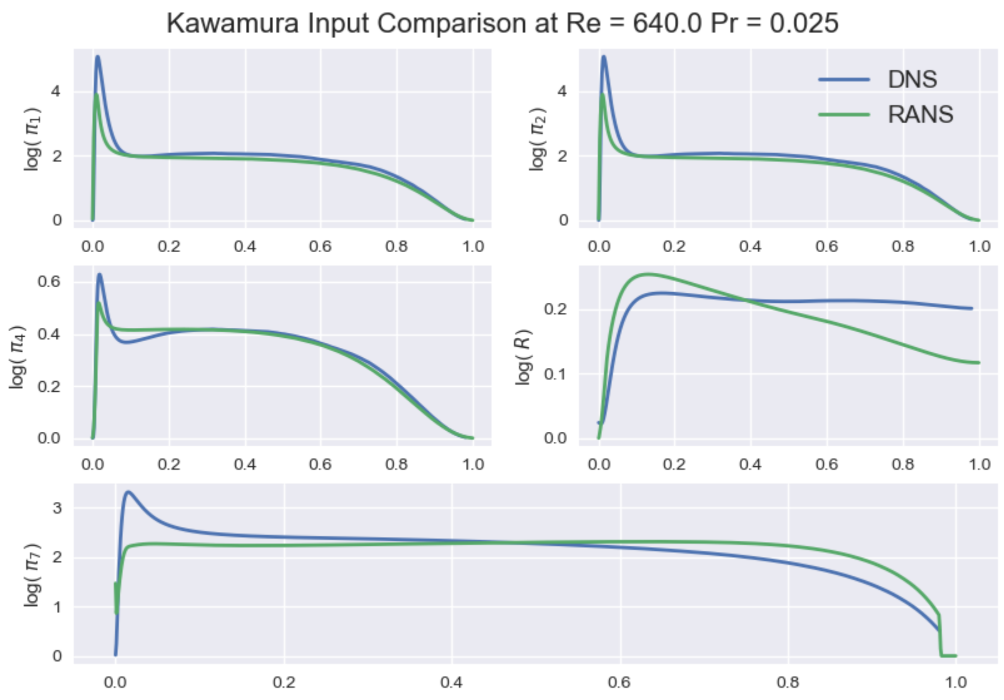


Figure 3.2: DNS and RANS inputs $(\pi_1, \pi_2, \pi_4, R, \pi_7)$ comparison.

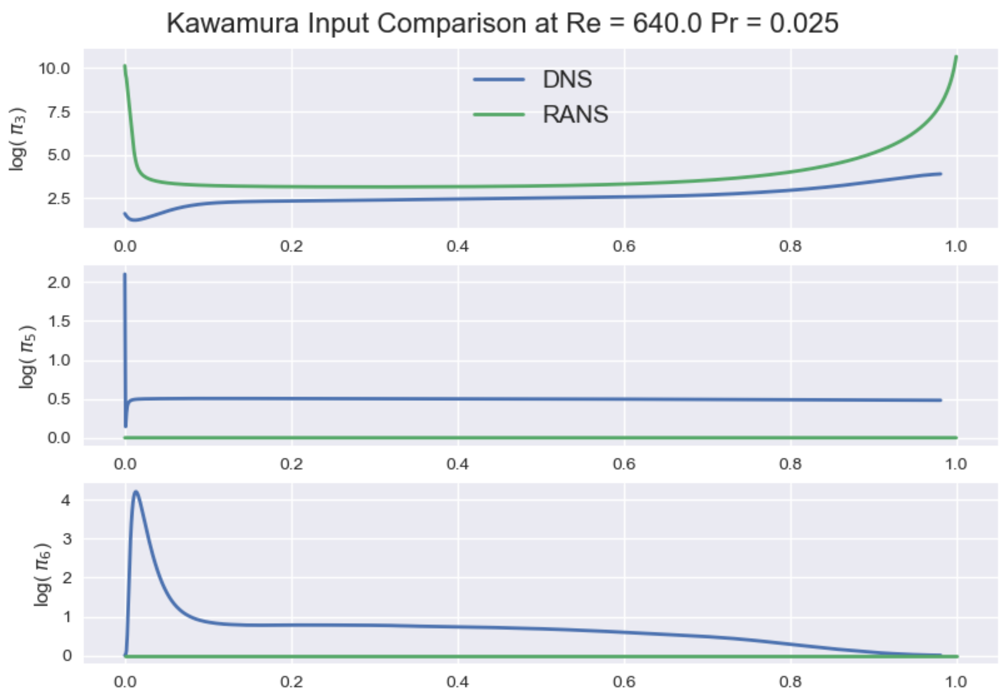


Figure 3.3: DNS and RANS inputs (π_3, π_5, π_7) comparison.

Backward Facing Step

The backward-facing step (BFS) is one of the simplest upgrades to the channel geometry. The DNS simulation for this test case was made by Oder et al. in [25] and the sketch of the geometry can be seen in Figure 3.4.

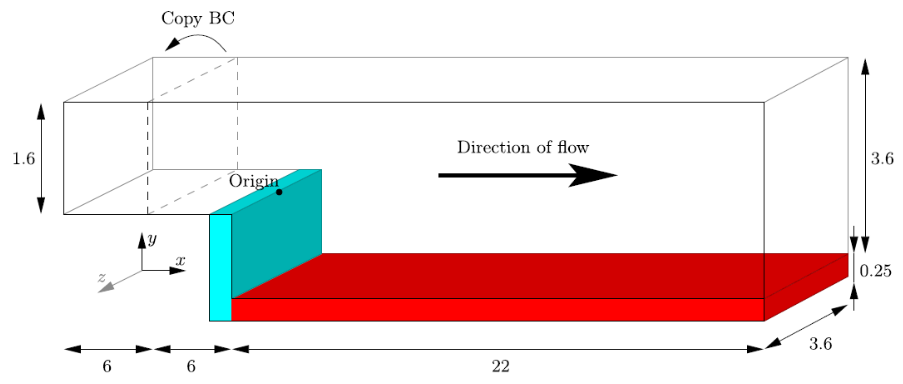


Figure 3.4: A representation of the BFS domain. The cyan part is the wall and the red one is the heater [25].

In the dimensionless unit the inflow is $12 \times 1.6 \times 3.6$ and the outflow is $22 \times 3.6 \times 3.6$. Considering a length unit $h = 25\,\mathrm{mm}$ the inflow has dimensions of $40\,\mathrm{mm} \times 90\,\mathrm{mm}$ and the outflow of $90\,\mathrm{mm} \times 90\,\mathrm{mm}$.

The inverse viscosity, marked with Re_b is calculated from the mass flow rate as:

$$\mathrm{Re}_b = \frac{\dot{m}}{L_z L_y \mu} h \tag{3.2}$$

where $\dot{m}$ is the mass flow rate thought the domain, L_z and L_y are the dimensional width and height of the inflow and μ is the dynamic viscosity. The viscosity in this case is constant and does not depend on the temperature. The inverse viscosity Re_b is connected to the Reynolds number based on the hydraulic diameter of the inflow and the bulk velocity as shown in Equation 3.3.

$$\mathrm{Re} = \frac{2 L_y L_z}{h(L_y + L_z)} \mathrm{Re}_b \tag{3.3}$$

In the simulation conducted the inverse viscosity was equal to $\mathrm{Re}_b = 3200$ and the Prandtl number was $\mathrm{Pr} = 0.005$ that corresponds to the sodium Prandtl number and $\mathrm{Pr} = 0.1$ which corresponds to the upper limit of Prandtl numbers of liquid metals.

In Figure 3.5 are represented the inputs for $x = 2$.

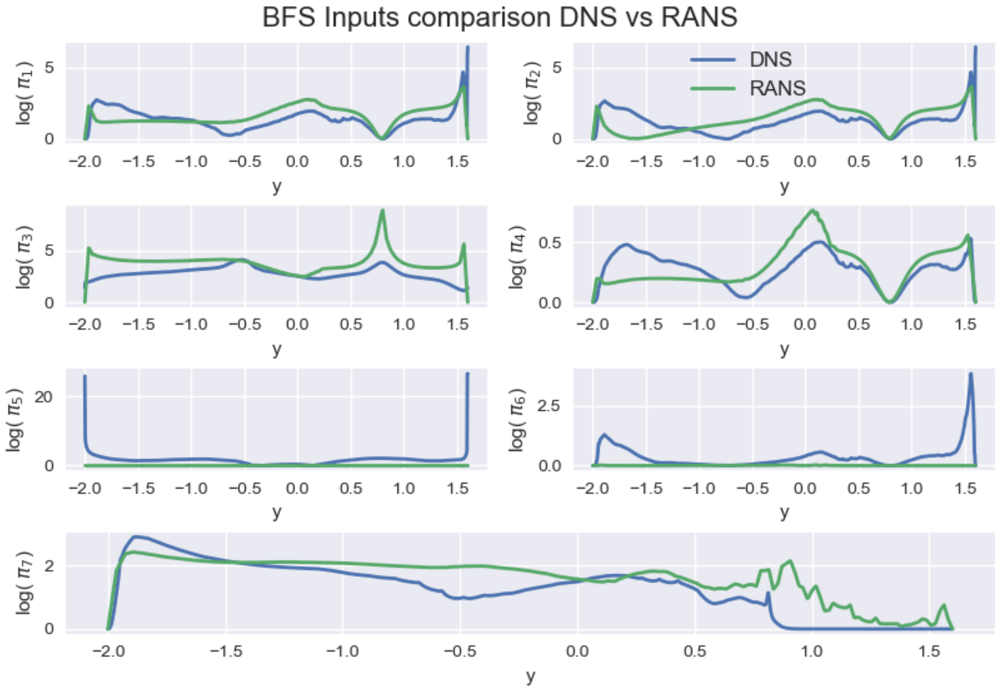


Figure 3.5: Backward facing step inputs representation at $x = 2$.

It can be seen in Figure 3.5 that, π_5 and π_6 are very different from the DNS and the RANS cases. This is caused by Boussinesq's hypothesis applied in the $k - \epsilon$ momentum turbulence model demonstrated in Appendix A.

3.2 Input space visualisation

It is easy to observe that since the input features are 10, a 10-dimensional space will be created and more specifically, a 10 dimensions hypercube. Taking a closer look at the formulation of the inputs from Table 2.2, it can be seen that, for example, π_1 and π_2 are mostly related to the momentum field ($U,\ k,\ \epsilon$ are in the formulation), π_3 and π_5 are a function of the anisotropic part of the Reynolds stress tensor and, π_7 has the temperature quantity inside. It should be noted that the linear relationship between $\overline{u'\theta'}$ and ∇T, which is at the basis of the simple gradient diffusion hypothesis, will be altered since the gradient of the temperature is now a neural network input.

It turns out to be of fundamental importance to observe the space in which the inputs lie in order to observe first how this n-dimensional space is populated and secondly to understand if there are dimensions that collapse into vertices of this 10-dimensional hypercube.

To recap, each input of the neural network consists of 10 dimensions (π_1, π_2, π_3, ..., Pr). Therefore, only one scalar value for a single computational point will be obtained for each dimension.

A visualization of this kind is impossible to obtain with a simple plot since it would be limited only to the three dimensions so a parallel plot was made in order to see all the dimensions. This kind of plot is represented in Figure 3.6. Each individual line of the parallel plot is a computational point that has a different value for each input dimension and is connected to all the dimensions to show their numerical value.

A representation of the input space has been made in Figure 3.6 for the channel flow input in order to compare DNS (red line) and RANS (green line) inputs.

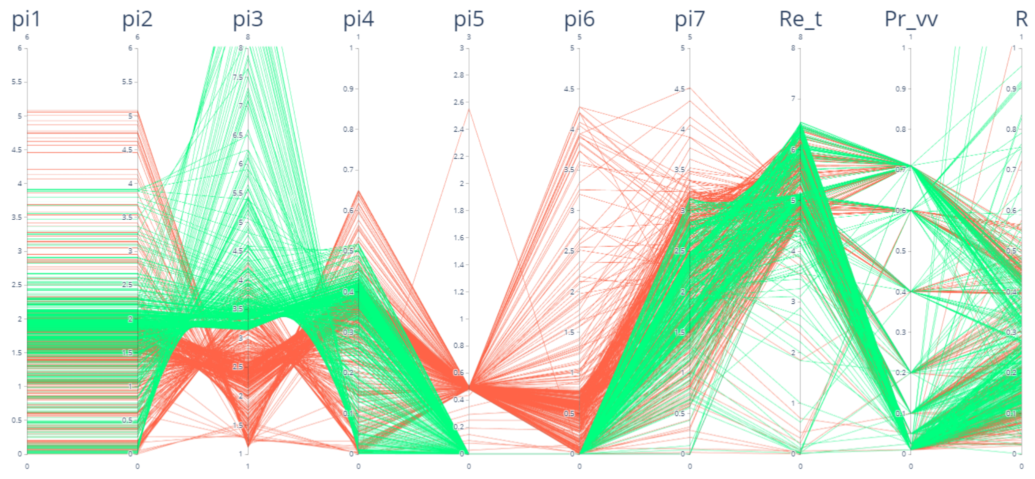


Figure 3.6: Parallel plot for 50 random samples for each channel flow simulation. Red lines are the DNS inputs and green lines are the RANS inputs.

From Figure 3.6 it can be noted that:

- π_1 and π_2 assume the same values for each samples. This is a particular behavior of the channel flow since is a 1D simulation and demonstrated in section A.2;
- π_3 is more dispersed with RANS inputs than DNS ones,
- π_5 is collapsing into a single point for both DNS and RANS inputs;
- π_6 is always zero for the RANS inputs, on the contrary, is highly dispersed for the DNS case (Figure 3.7);

- Pr_{vv} is always a certain value since all the samples come from a constant Prandtl number;
- The ratio R is strictly connected to the Prandtl number. For every Prandtl number exists a related range for R.

A dispersion analysis turns out to be very useful to see how datasets inputs are dispersed into the 10-dimensional hypercube.

3.3 Dispersion Analysis

This analysis was made in order to see if some dimensions collapse to a single point. In this analysis, the dispersion index was used. This index can be computed as follows:

$$\text{Dispersion index} = \frac{\sigma^2}{\mu} \tag{3.4}$$

where σ^2 is the variance and μ is the mean of the dataset.

The dispersion index or coefficient of dispersion is a normalized measure of the dispersion of a probability function. It's a powerful indicator that quantifies the clustering or dispersion of a set of observed occurrences relative to a standard statistical pattern.

The dispersion index chart is shown in Figure 3.7.

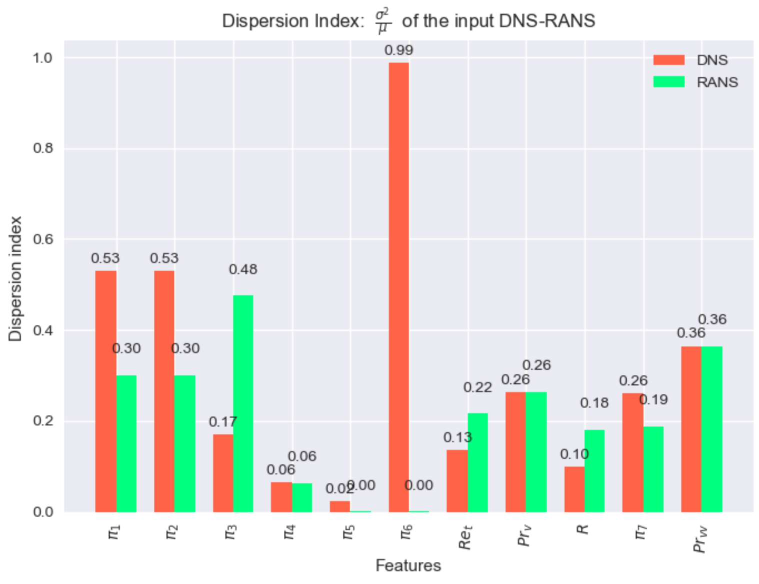
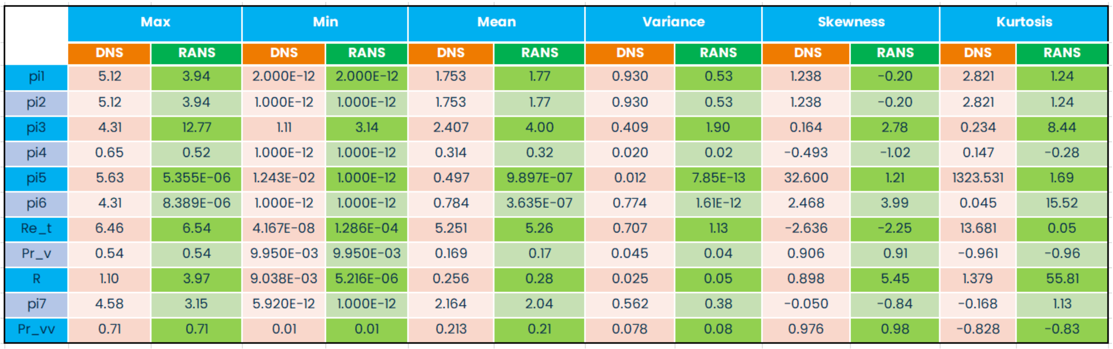


Figure 3.7: Dispersion index (computed in Equation 3.4) for the channel flow inputs.

From Figure 3.7 what can be observed is that:

- π_1 and π_2 are more dispersed in the DNS dataset and they have the same value due to the 1D approximation (section A.2);
- π_3 is more dispersed in the RANS dataset, this behavior can also be seen in Figure 3.6;
- π_5 and π_6 for the RANS dataset are collapsing into a single point for (section A.1);
- π_6 in the DNS dataset has a dispersion index $D \approx 1$ which means that it has similar variance and mean value (Poisson distribution);
- For the remaining inputs the dispersion is not zero which means that the dimension is not collapsing to a single point.

A statistical analysis of all the dimensions was made to analyze the statistical indexes. This analysis can be seen in Figure 3.8.



	Max		Min		Mean		Variance		Skewness		Kurtosis	
	DNS	RANS	DNS	RANS	DNS	RANS	DNS	RANS	DNS	RANS	DNS	RANS
pi1	5.12	3.94	2.000E-12	2.000E-12	1.753	1.77	0.930	0.53	1.238	-0.20	2.821	1.24
pi2	5.12	3.94	1.000E-12	1.000E-12	1.753	1.77	0.930	0.53	1.238	-0.20	2.821	1.24
pi3	4.31	12.77	1.11	3.14	2.407	4.00	0.409	1.90	0.164	2.78	0.234	8.44
pi4	0.65	0.52	1.000E-12	1.000E-12	0.314	0.32	0.020	0.02	-0.493	-1.02	0.147	-0.28
pi5	5.63	5.355E-06	1.243E-02	1.000E-12	0.497	9.897E-07	0.012	7.85E-13	32.600	1.21	1323.531	1.69
pi6	4.31	8.389E-06	1.000E-12	1.000E-12	0.784	3.635E-07	0.774	1.61E-12	2.468	3.99	0.045	15.52
Re_t	6.46	6.54	4.167E-08	1.286E-04	5.251	5.26	0.707	1.13	-2.636	-2.25	13.681	0.05
Pr_v	0.54	0.54	9.950E-03	9.950E-03	0.169	0.17	0.045	0.04	0.906	0.91	-0.961	-0.96
R	1.10	3.97	9.038E-03	5.216E-06	0.256	0.28	0.025	0.05	0.898	5.45	1.379	55.81
pi7	4.58	3.15	5.920E-12	1.000E-12	2.164	2.04	0.562	0.38	-0.050	-0.84	-0.168	1.13
Pr_vv	0.71	0.71	0.01	0.01	0.213	0.21	0.078	0.08	0.976	0.98	-0.828	-0.83

Figure 3.8: Statistical indexes for the channel flow test case in the DNS and RANS dataset.

As can be seen in Figure 3.8, π_5 has a strong skewness and Kurtosis, leading to the assumption that this type of distribution is not a normal distribution. In addition, a skewed dataset means that the distribution of data is not balanced or symmetrical. This fact can cause problems in the training phase of the model since the model is also sensitive to data distribution. Let's imagine a dataset with a skewed distribution toward a value, the model, in this case, may tend to overemphasize the predicted value toward the skewed data. High oscillations around the mean are observed for π_1 and π_2 since they have the largest value of variance.

A principal Component Analysis (PCA) on the dataset was performed in order to, first interpret the two datasets and after to reduce their dimensionality.

3.4 Principal Component Analysis (PCA)

This section firstly describes the methodology used to carry out a Principal Component Analysis (PCA) and secondly describes how it was applied to the mixed dataset (DNS, RANS) to observe whether correlations were created between features and to reduce their dimensionality.

3.4.1 Introduction

Existing datasets have more and more features. The interpretation of these datasets requires methods that are able to reduce their dimensionality while preserving most of their information. Therefore many techniques have been developed to accomplish this problem and Principal Component Analysis (PCA) is one of the oldest and most widely used. The idea is simple: reduce the dimensionality of the dataset while preserving the variability as much as possible [26].

Preserving as much variability means finding variables, which are linear functions of features in the original dataset that maximize the variance and are uncorrelated with each other. Finding these variables leads us to solve an eigenvalues and eigenvectors problem [26].

Pearson K. (1901) [27] and Hotelling H. (1933) in [28] are the first in the literature who talk about PCA. In more recent times, entire books have also been written about many possible variants developed of the algorithm [29, 30].

In subsection 3.4.2 it can be seen the formal definition of the PCA which can be achieved as the solution to the eigenproblem or, alternatively, from the Singular Value Decomposition (SVD) of the data matrix.

3.4.2 Methodology

The data matrix consists of $n \times p$ entries where n is the number of samples/entities/individuals and p is the number of features/variables (e.i. π_i groups in our case). This data matrix can be called $\boldsymbol{X}$, and it is thus necessary to seek a linear combination between the column of the matrix $\boldsymbol{X}$ that accomplishes the maximum variance.

$$\boldsymbol{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{11} & x_{12} & \cdots & x_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}$$

This linear combination can be written in Equation 3.5 following [26]:

$$\boldsymbol{X}\boldsymbol{a} = \sum_{j=1}^{p} = a_j \boldsymbol{x}_j \tag{3.5}$$

where **a** is a vector of constants $a_1, a_2 \ldots, a_p$. In this case, the variance of any linear combination is given by:

$$\text{var}\,(\boldsymbol{Xa}) = \boldsymbol{a'Ka} \tag{3.6}$$

where **K** is the sample covariance matrix associated with the dataset and ' denotes transpose. The aim is to obtain a p-dimensional vector **a** that maximizes **a'Ka** with the constrain that **a'a**=1 for the reason that the problem should be well-defined [26].

The problem is equivalent to maximizing:

$$\mathbf{a'Ka} - \lambda(\mathbf{a'a} - 1) \tag{3.7}$$

where λ is the Lagrange multiplier. To maximize Equation 3.7 a differentiation with respect to **a** and the equalization with the null vector should be done:

$$\boldsymbol{K}\mathbf{a} - \lambda\mathbf{a} = 0 \iff \mathbf{Ka} = \lambda\mathbf{a} \tag{3.8}$$

As a consequence of Equation 3.8 for the definition [1]: **a** must be a unit-norm eigenvector, λ will be the eigenvalue of the covariance matrix **K**. From Equation 3.6 and 3.8 can be obtained what follows:

$$\text{var}\,(\boldsymbol{Xa}) = \boldsymbol{a'Ka} = \lambda\boldsymbol{a'a} = \lambda$$

It's clear that it is interested in the highest eigenvalue (λ_1) and the corresponding eigenvector ($\boldsymbol{a_1}$) to maximize the variance.

A real symmetric matrix $p \times p$, such as **K**, has exactly p real eigenvalues $\lambda_k(k = 1, \ldots, p)$. The corresponding eigenvectors are established to make an orthonormal set of vectors. In this case, it is obtained a set of eigenvectors with a maximum of p linear combinations $\boldsymbol{Xa} = \sum_{j=1}^{p} a_{jk}\boldsymbol{x_j}$ that maximize the variance with the constrain to be uncorrelated with the previous linear combinations [26].

The uncorrelation feature can be explained since the eigenvectors are made with an orthonormal set e.g. $\boldsymbol{a'_k a_{k'}} = 1$ if $k = k'$ and zero otherwise. Writing the covariance for the two possible linear combinations:

$$\text{cov}\,(\boldsymbol{Xa_k}, \boldsymbol{Xa_{k'}}) = \boldsymbol{a'_{k'}Ka_k} = \lambda_k\boldsymbol{a'_{k'}a_k} = 0 \qquad \text{if } k' \neq k \tag{3.9}$$

the uncorrelation between two different linear combinations is demonstrated.

$\boldsymbol{Xa_k}$ combinations are the so called *principal components* of the dataset. The elements of $\boldsymbol{a_k}$ vector are called in the common terminology *PC loadings* and the elements of the linear combination $\boldsymbol{Xa_k}$ are called *PC scores*.

[1]Eigenvectors (v_i) of the matrix **A** are specified as $\mathbf{A}v_i = \lambda_i\ v_i$ with λ_i the eigenvalues

PCA is a dimensionality reduction method since an original set of variables p can be replaced by q derived optimized variables. For example, if one wants to represent the dataset in a more "understandable" way $q = 2$ or $q = 3$ is preferred since can be easily plotted. In order to understand better how many dimensions can be reduced without losing too much variance a parameter can be defined to measure the quality of a given j-principal component as follows:

$$\pi_j = \frac{\lambda_j}{\sum_{j=1}^{p} \lambda_j} = \frac{\lambda_j}{\{\boldsymbol{K}\}} \tag{3.10}$$

where $\{\boldsymbol{K}\}$ is the trace of the covariance matrix which is the measure of the total variance. The fact that the variance can be summed allows talking about the proportion of total variance explained by a set of PC components. A common cut-off is a threshold of 70% of the total variance. It means that the first q PCs components should be found to accomplish together this value.

3.4.3 PCA algorithm in the Channel Flow Dataset

The PCA algorithm described above in subsection 3.4.2 was applied to the channel flow (1D flow) in the mixed dataset with DNS and RANS data. This mixed dataset can be visualized in Figure 3.6 in all of its dimensions, and the goal is to try to reduce its dimensionality from 10 dimensions to a size that can represent at least 70% of the variance.

Mixed Database Size for 1D channel flow

17 DNS simulations × 1000 computational points	+
17 RANS simulations × 1000 computational points	=
34 TOTAL simulations × 1000 computational points	

Table 3.2: The table shows the size of the channel flow 1D database. It is clear that the size of the database is composed of 34 000 computational points coming from the 17 DNS and 17 RANS simulations with specific Reynolds and Prandtl numbers according to Table 3.1.

From Table 3.2 is evident that the computational points for each simulation are 1000, resulting in 34 000 points with all the 34 simulations.

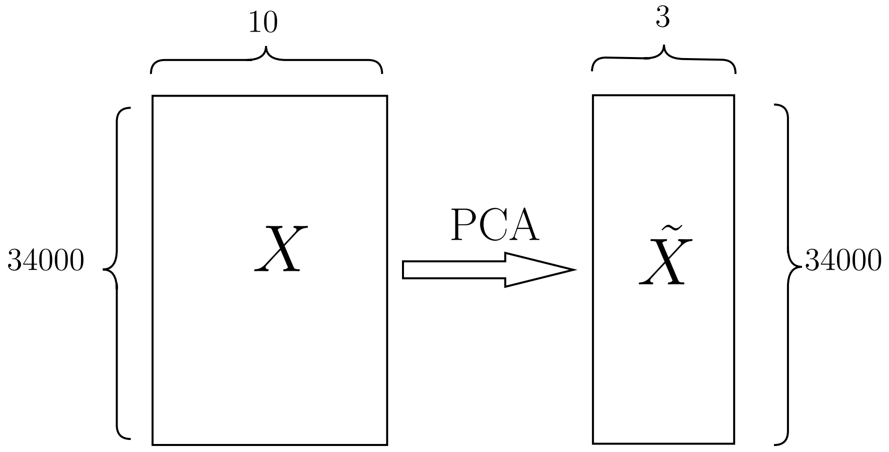


Figure 3.9: Input dimensional reduction from 10 dimensions to 3

Spatial reduction, in this case, can be performed with the following method:

1. Standardization of the dataset to remove the mean and to scale to unit variance. The formula applied is $X' = \frac{X-\mu}{\sigma}$ where X' is the scaled dataset, X is the old dataset, μ is the mean and σ is the standard deviation of the dataset. Standardization is a common technique for many machine learning estimators: they can behave badly if the individual features don't look like a normal distribution. In this case was applied the *StandarScaler* function from scikit learn;

2. Computing the covariance matrix of the dataset;

3. Computing the eigenvectors and eigenvalues;

4. Choosing the q new dimensions that one want to keep, and creating the transformation matrix;

5. Transform the original dataset into the new one with reduced dimensions.

The steps from 2 to 5 are done in the *PCA* algorithm in the scikit learn package.

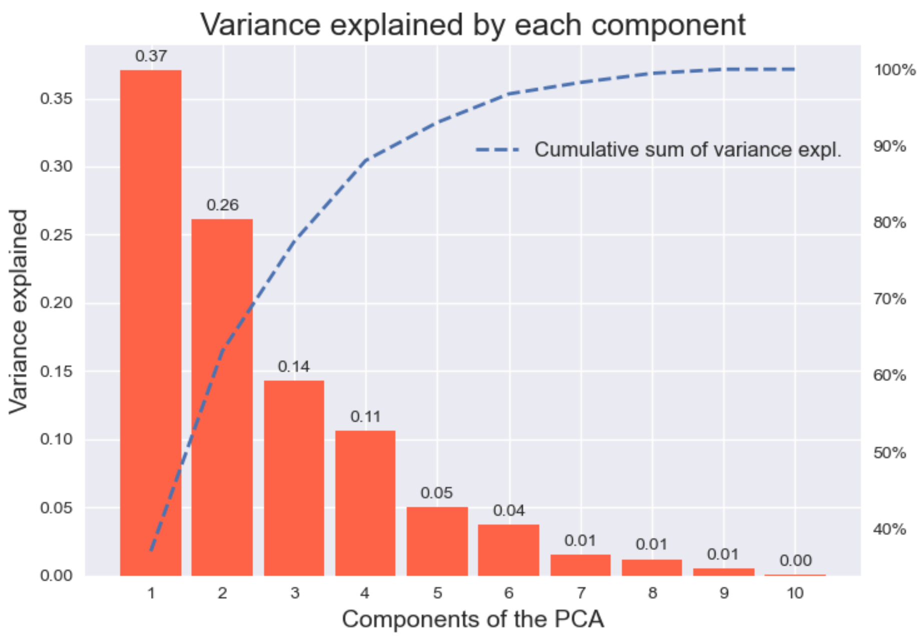


Figure 3.10: Variance explained by each component of the PCA algorithm. In the bar plot, the variance explained by each new dimension of the decomposition can be observed. The dotted blue line shows the cumulative sum of the variance explained.

In Figure 3.10 is shown the variance explained for each component. It's clear that if only 3 components are chosen (PC1, PC2, PC3) only the 77.5% of the total variance can be explained, according to Equation 3.10. 77.5% of the total variance is a good value, allowing it to represent the new dataset in a more understandable way. The first three components were chosen to represent the reduced datasets and in Figure 3.11 it is represented how each one of these new component explains the variance of every single feature.

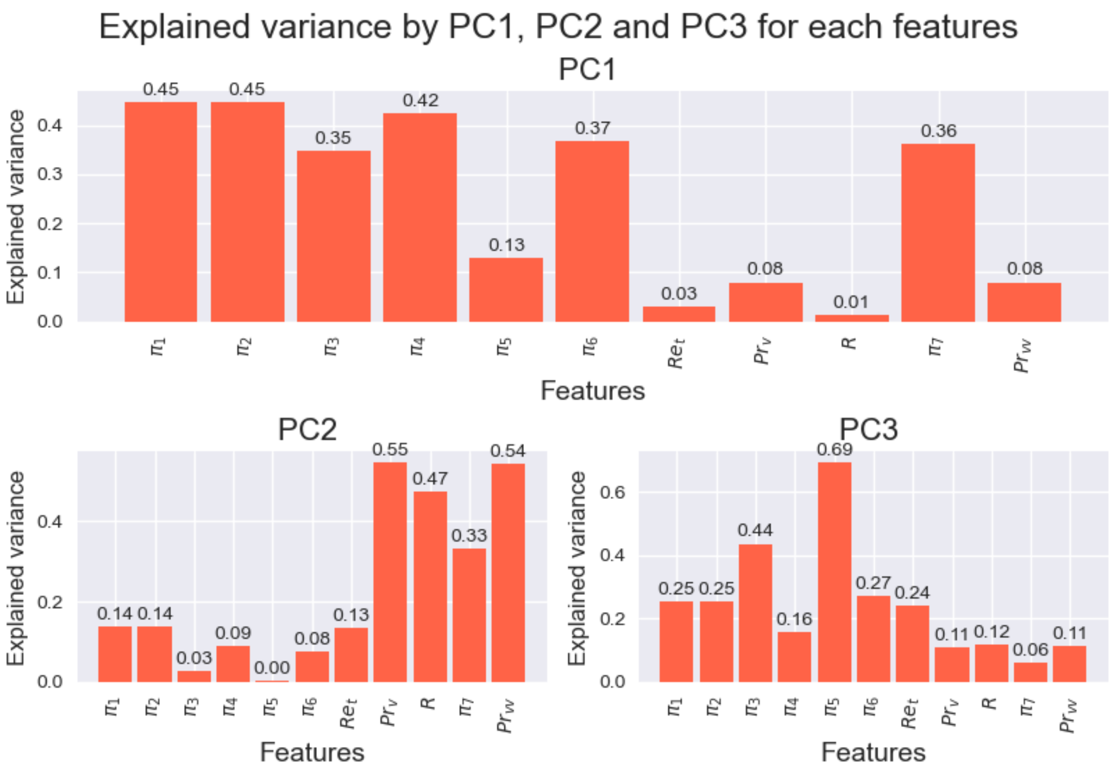


Figure 3.11: Explained variance for each principal component for each feature. On the top PC1 explains most of the inputs related to the momentum part, on the bottom left PC2 that explains most all the features related to the thermal part, and on the bottom right PC3 explains mostly all the features related to the closure term (the one that differs from DNS to the RANS datasets).

PC1	**PC2**	**PC3**
$\pi_1 = 45\%$	$Pr = 55\%$	$\pi_5 = 69\%$
$\pi_2 = 45\%$	$R = 47\%$	$\pi_3 = 44\%$
$\pi_4 = 42\%$	$\pi_7 = 33\%$	$\pi_6 = 27\%$
$\pi_6 = 37\%$	...<30%	...<25%
$\pi_7 = 36\%$	...<30%	...<25%
...<33%	...<30%	...<25%

Table 3.3: Features variance explained by each principal component.

Table 3.3 represent what was previously observed in Figure 3.11 in a tabular way. From Figure 3.11 and Table 3.3 can be stated what follows:

1. Only three components are able to explain the 77.5% of the total variance

of the dataset which is a good number to enhance the interpretability and the visualization. This also means that the original set of input features was redundant;

2. The first Principal Component (PC1), according to Table 2.2, explains most of the features related to the momentum part such as i.e. U, k, ϵ and the mean strain rate;

3. The second Principal Component (PC2), according to Table 2.2, explains most of the features related to the thermal part such as i.e. Pr, R, π_7;

4. The third Principal Component (PC3), as seen in Table 2.2, explains most of the features that differ from the DNS and the RANS frameworks i.e. π_3, π_5, which are functions of the anisotropic part of the Reynolds stress tensor.

A visualization of the complete dataset can be made for the channel flow test case with the new base obtained from the PCA analysis, which is shown in Figure 3.12

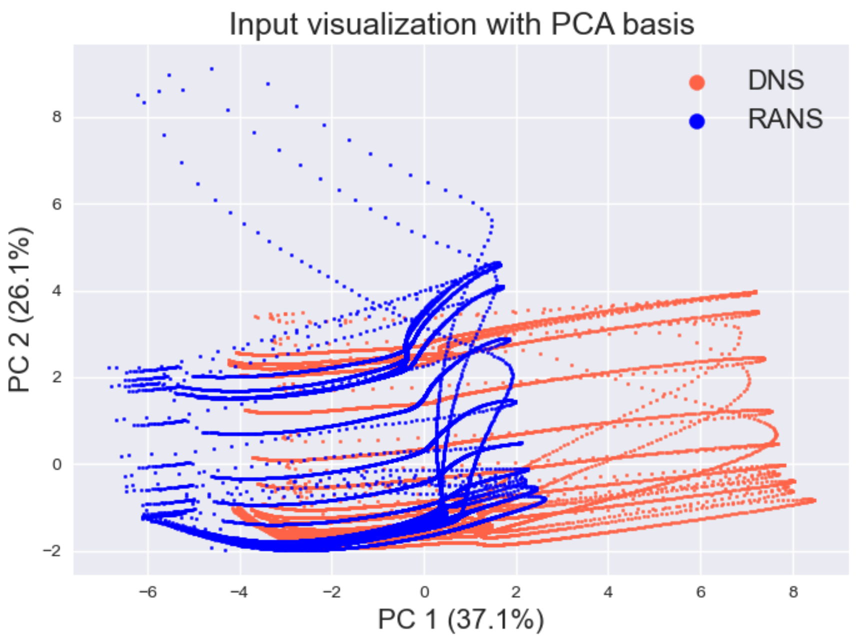


Figure 3.12: Representation of the full channel flow database (Table 3.2) in the PCs axes. Can be stated clearly that the two databases differ mostly in the PC1 axes instead of the PC2. This behavior is due to the fact that the RANS thermal part (the one mostly explained by PC2) is the same in the DNS and RANS framework as told in section 3.1.

In Figure 3.12 is shown a representation of PC1 and PC2 of the full database, the one mentioned in Table 3.2. As can be seen, the points computed with the DNS database differ a lot in the PC1 axis instead of the PC2 axis. This is due to the fact that PC2 is mostly related to the thermal features and these features are the same in the DNS and RANS framework as told in section 3.1 for the construction of the database.

In order to enhance the interpretability of the explained components, a plot with only one single point of each simulation (34 in total) has been made which relates the first and the third principal components. A representation of this chart was made in Figure 3.13, where it's easy to observe a broad division from DNS and RANS computed inputs. This division is made by π_3 and π_5 features that are explained by the PC3 component. These two features are the ones that, as written before, are very different from the DNS and the RANS database, in addition, they are the ones that can allow the network to detect what kind of momentum model has been applied and improve its compatibility with a thermal model. Looking at it in another way; it is like having a classifier inside of the network that knows exactly, just looking at π_3 and π_5, in which starting conditions it is.

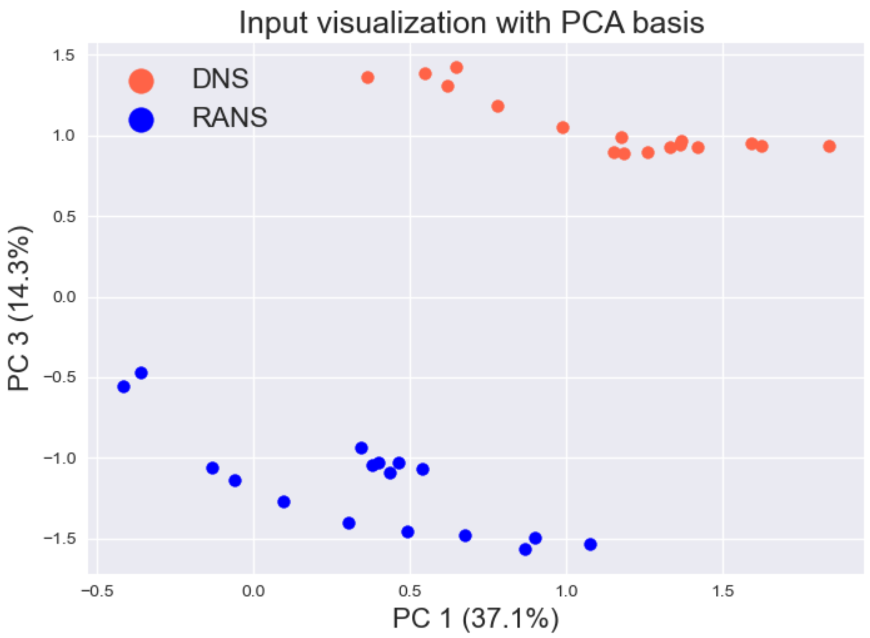


Figure 3.13: Representation of just one point for each channel flow simulation (34 in total) for the first and the third principal components. A huge division can be observed since PC3 is mostly related to π_5 and π_3.

3.5 Input Analysis conclusion

In this chapter, the inputs from DNS and RANS simulations were computed, analyzed, and compared. In section 3.1 it was observed that some inputs changed a lot between DNS and RANS datasets. In particular, these inputs were π_3 and π_5 and π_6, as it can be noted in Figure 3.6.

After visualizing the 10-dimensional space in which the inputs lie, a PCA analysis was performed in order to try to reduce the dimensions of the input for better interpretability of the input space.

By observing the PCA analysis, the following can be determined:

- With only 3 components (PC1, PC2, PC3) can be explained the 77.5% of the total variance. This means that performing a training of this model with a reduced order of inputs, 22.5% of the variance of the dataset will be lost;
- The new principal components were able to "cluster" the old ones since it was observed that the first principal component is more related to all the momentum inputs (depending mostly on k, ϵ); the second principal component is mostly related to the thermal part (Pr, R, π_7) and the third component is the one that is composed by all the therm that differs a lot from the DNS and the RANS framework;
- It is due to the third component that one can state to be able to train a model that can perform well with both inputs

Now that the input space has been observed, it is necessary to proceed to train the model. To do this, the diversity of the two datasets studied above must be taken into account. In addition, it must be also considered how to evaluate the loss function previously written in Equation 2.26 in the best way.

Chapter 4

Multi-objective optimization

Multi-objective optimization is a well-known technique for dealing with complex optimization problems involving conflicting variables. In real applications, sometimes, it is not possible to optimize a single target without others being affected. That is why the multi-object optimization technique has become widely used. An excellent technique to visualize this trade-off between two or multiple possible objectives is the Pareto front. The Pareto front, therefore, can be seen as a map in which all possible solutions are depicted that can no longer optimize one objective without sacrificing the performance of the others. The Pareto Front thus manages to provide an intuitive and clear way to help the decision-maker choose the optimal way.

This chapter focuses on the application of multi-objective optimization to the redesign of the network and especially the loss function previously used and shown in Equation 2.26. In particular, the focus will go firstly on the formulation of the loss function and secondly on its minimization with the two conflicting objects (DNS and RANS inputs).

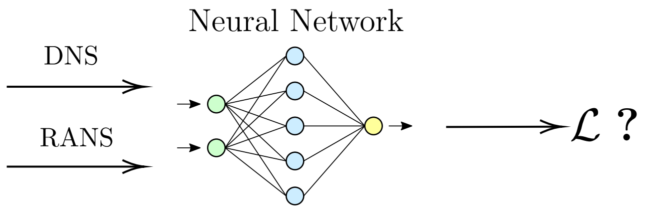


Figure 4.1: The above figure shows the problem of the mixed inputs computing the loss function ($\mathcal{L}$) that will be faced in section 4.1.

4.1 Trade-Offs and Considerations in Loss Function Design

Since the goal of the neural network is to manage input data of different levels of accuracy, the loss function should be designed to account for the multi-objective optimization of both DNS and RANS inputs. To achieve this, a multi-objective loss function is proposed which is based on multiple terms. Recalling the loss function formulation of Equation 2.26 and generalizing to a generic mini-batch (b):

$$\mathcal{L}_b = \frac{1}{N}\left(\sum_{i=1}^{N}\sum_{j=1}^{3}\left(\hat{q}_{i,j} - q_{i,j}\right)^2\right)\Bigg|_b + \frac{\lambda}{N}\left(\sum_{i=1}^{N}\sum_{j,k=1}^{3}\left|\frac{\partial \hat{q}_{i,j}}{\partial x_k} - \frac{\partial q_{i,j}}{\partial x_k}\right|\Delta x_k\right)\Bigg|_b \tag{4.1}$$

In agreement with what was said in section 3.1 the database is constituted by DNS and RANS inputs both 1D (from the channel flow simulation) and 2D (from the backward facing step simulation). The loss function can be written in the following way:

$$\mathcal{L} = \underbrace{\mathcal{L}_{1DNS} + \mathcal{L}_{2DNS}}_{\mathcal{L}_{DNS}} + \alpha\left(\underbrace{\mathcal{L}_{1RANS} + \mathcal{L}_{2RANS}}_{\mathcal{L}_{RANS}}\right) \tag{4.2}$$

where α is a hyperparameter that should be tuned and represent the weight of the DNS and RANS losses. In fact, if α is very small the optimizer will minimize only the DNS losses and, on the contrary, if α is very high, only the RANS losses would be minimized.

The goal is to obtain the best α value that minimizes both DNS and RANS losses. To achieve this, many trainings must be carried out with different α values. This operation will be computationally intensive since the network should be trained with 7 different values for α and, at least 30 times for statistical purposes. For this reason, the network was trained through the graphics card (GPU), reducing the computational time by more than six times as seen in Table 4.1.

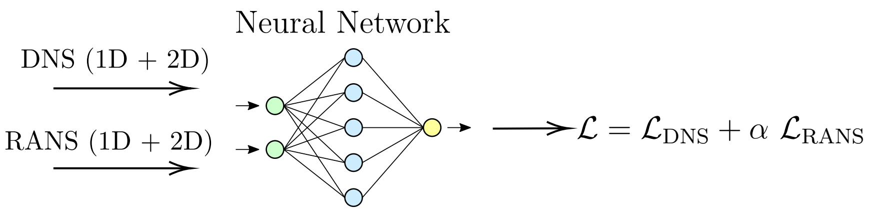

Neural Network
DNS (1D + 2D)
RANS (1D + 2D)
$\mathcal{L} = \mathcal{L}_{\text{DNS}} + \alpha\ \mathcal{L}_{\text{RANS}}$

Figure 4.2: In this figure a schematic view of the training process can be seen and how the new loss function is being computed. This picture will be more completed and further recalled.

4.2 GPU acceleration in Machine Learning

The more the complexity and size of the network grow, the training can become more and more prolonged and intense. To address this problem, graphics cards (GPUs) emerged, which enabled faster trainings and better energy efficiency for them.

Firstly is necessary to know what is a CPU and how it works. The CPU or central processing unit is the primary processing component of a computer. The CPU is a key element in the computer architecture since is responsible to execute instructions that allow its working. The classic job of the CPU is to receive data from the computer's RAM memory, process them with logical and/or arithmetic operations, and finally save the results back into memory. The CPU is classified as the "brain" of the computer since is responsible for most of the computer's tasks. Traditionally, CPUs were single-core but nowadays they are multi-core with, in general, more than 2 cores and less than 32. This multicore architecture can allow performing tasks in a parallel way which will be a key element in the neural network training.

The GPU or graphics processing unit is a computer processor that is designed to work with complex graphics calculations. The GPU is a high-performance processor that is able to execute multiple calculations in parallel, making this well-suited to accelerate tasks such as neural network training.

The reason why GPUs are more used in machine learning is due to the memory bandwidth. First of all, CPUs are latency-optimized and GPUs are bandwidth optimized they can be visualized as a CPU being a fast racing car and GPU as a huge truck. They both have to transfer packages between points A and B, the CPU (racing car) will transport fewer packages but in a very fast way, and the GPU (the truck) will carry a higher number of packages but very slowly. However, the CPU needs to go back and forth many times to do the same job as the GPU. In order words, CPUs are really good to fetch a small amount of memory in a fast way (i.e. doing $2 \times 3 \times 6$) and GPUs are really good to fetch a large amount of memory (i.e. matrix multiplication like $\boldsymbol{A} \times \boldsymbol{B} \times \boldsymbol{C}$)

$$\begin{array}{ll} \text{With CPU} & A \xrightarrow{\text{2X}} B \\ \text{With GPU} & A \xrightarrow{\text{100X}} B \end{array}$$

It's clear that GPUs carry more data at the same time but the problem still remains if the transport is too slow due to the fact that the "truck" will need to come back every time. To fix this problem "thread parallelism" can be used. It means using a fleet of "trucks" that can carry a lot of memory that leaves at different times. In this case, one has to wait a little bit only for the first truck. Thread parallelism can in this way hide the high latency of the GPU and at the same time allows it to have high bandwidth.

Is then evident that in the machine learning framework, big matrix multiplications are to be faced, proceeding with GPU instead of CPU will be faster and this has been proved on the following charts made varying the neural network neurons number.

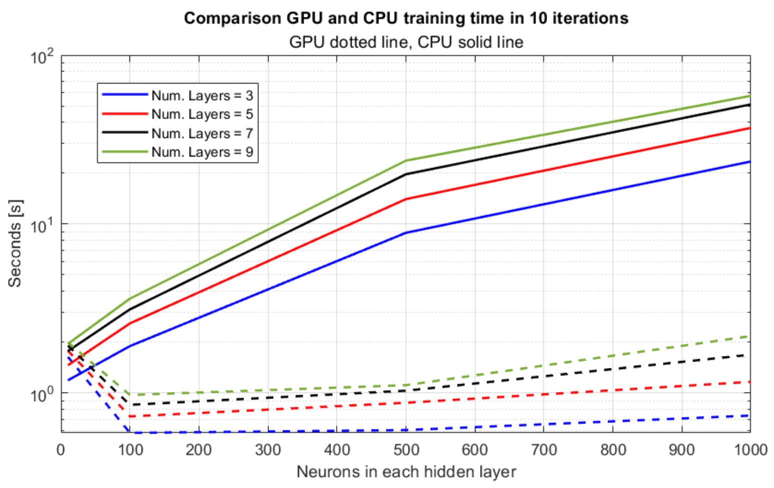


Figure 4.3: Computational time of 10 iterations (epochs) with a different number of layers and neurons each hidden layer with a CPU and GPU architecture.

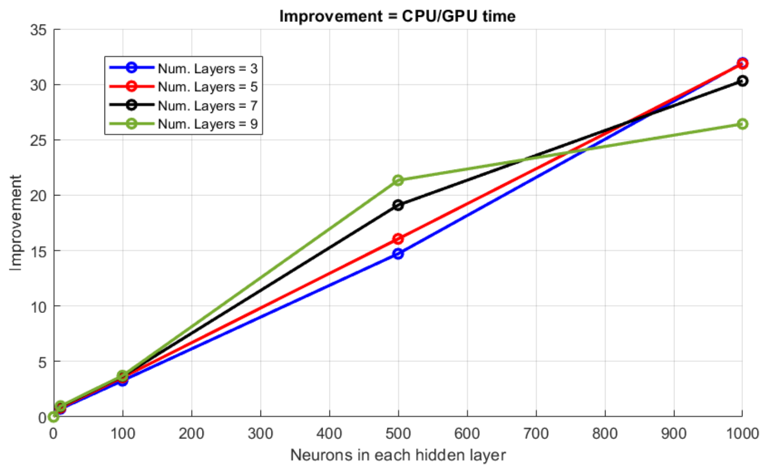


Figure 4.4: GPU improvement time with the number of layers and neurons. It's clear from this figure that with the bigger architecture, the computational time can be boosted of 25×.

In Figure 4.3 and 4.4 can be seen clearly that for larger networks the training with the GPU can be 25 times faster than the CPU. With a low number of neurons, the computational time of the CPU and GPU are mainly the same. To create this chart the GPU used was the NVIDIA 3090 TI and the CPU used had 16 cores.

The previous chart was made with a simple test case in order to compare the improvement made with the GPU instead the CPU. Table 4.1 will show the improvement that was obtained in the network architecture of Figure 2.3.

GPU vs CPU performances in the network

Architecture Type	**Single Training Time**	**Pareto Front Training Time**
CPU 32 Threads	253 min [$\approx$ 4.23h]	$\approx$ 50 days
NVIDIA RTX 3090 Ti	40 min [$\approx$ 0.66h]	$\approx$ 8 days

Table 4.1: Training time for the network in the two different architecture structures for a single training and for the Pareto front that is composed of 280 total trainings

The speed up in the training phase has been critically important for tuning the α parameter through the Pareto front. This made it possible to conduct many more trainings and enhance the statistical analysis.

4.3 Pareto Front

In this section, the discussion covered in section 4.1 will be continued. The main goal of this analysis is to perform a multi-objective optimization to get the best choice on the α parameter.

Firstly a vector with 7 values of α was created. This vector is the following:

$$\alpha = [0.001,\ 0.01,\ 0.1,\ 1,\ 10,\ 100,\ 1000] \tag{4.3}$$

initialized in this way to explore possible combinations of the α weight of Equation 4.2. The procedure to obtain the Pareto front shown in Appendix B can be summarized as:

1. Select α_k in Equation 4.3;
2. Compute the four forward pass $\boldsymbol{Y}_i^t = \mathcal{M}(\boldsymbol{X}_i, \mathcal{W}^t)$ as shown in Figure 4.5;
3. Evaluate the loss function according to Equation 4.2 ($\mathcal{L}^t$);

4. Change the weights and the biases $\mathcal{W}^{t+1}$ of the network in order to minimize the loss function $\mathcal{L}^t$
5. Repeat steps 2-4 for each epoch and save $\mathcal{L}^t_{\text{DNS}}$ and $\mathcal{L}^t_{\text{RANS}}$
6. Take the mean of the last 400 values of the DNS and RANS losses for each α: $\mathcal{L}_{\text{DNS},k}$ and $\mathcal{L}_{\text{RANS},k}$ for each α and create the Pareto front seen in Figure 4.6 (left);
7. Repeat steps 1-6 for 30 times for statistical purposes and computes the centroid of the points plotted in step 6. The results can be seen in Figure 4.6 (right).

The training procedure (2-4) is composed as described in subsection 2.2.2 with an ADAM optimizer, with a constant learning rate of 5×10^{-4} and a weight decay of 1×10^{-5} for 30 epochs and a batch size of 400.

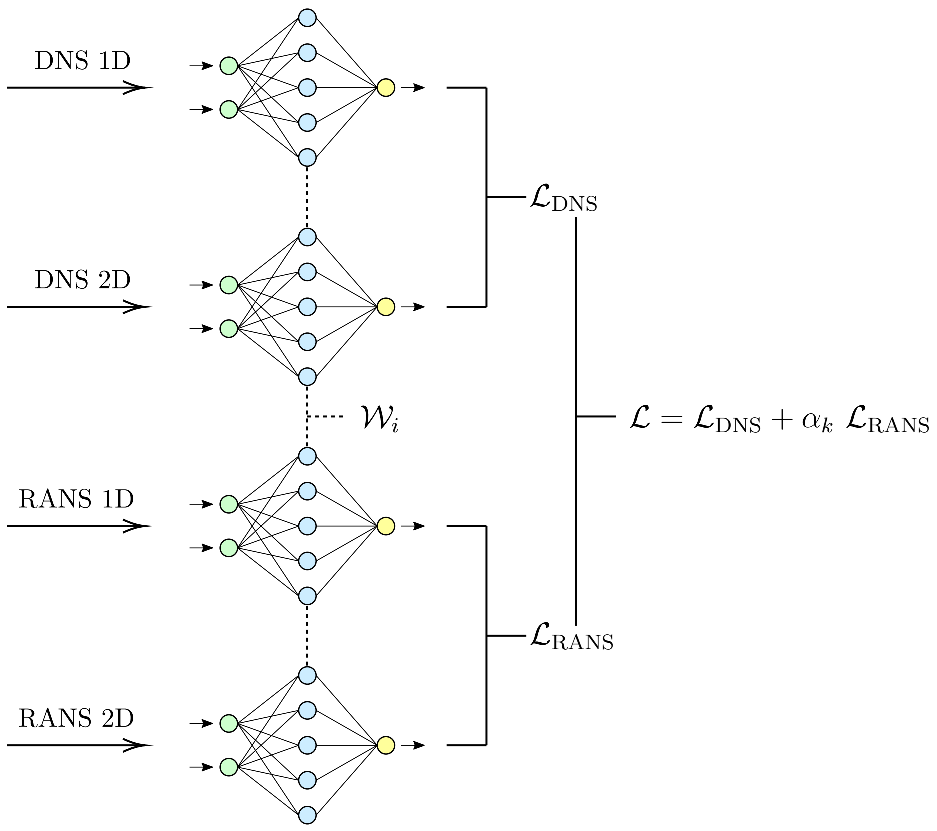

DNS 1D
DNS 2D
RANS 1D
RANS 2D
$\mathcal{L}_{\text{DNS}}$
$\mathcal{L}_{\text{RANS}}$
$\mathcal{W}_i$
$\mathcal{L} = \mathcal{L}_{\text{DNS}} + \alpha_k\ \mathcal{L}_{\text{RANS}}$

Figure 4.5: In this Figure is represented a schematic view of the training process. For every α_k selected from Equation 4.3, four forward passes and the loss function $\mathcal{L}$ have been computed. The value of $\mathcal{L}_{\text{DNS}}$ and $\mathcal{L}_{\text{RANS}}$ are computed in accordance with Figure 2.4 and Equation 2.26 for each α. Results in Figure 4.6.

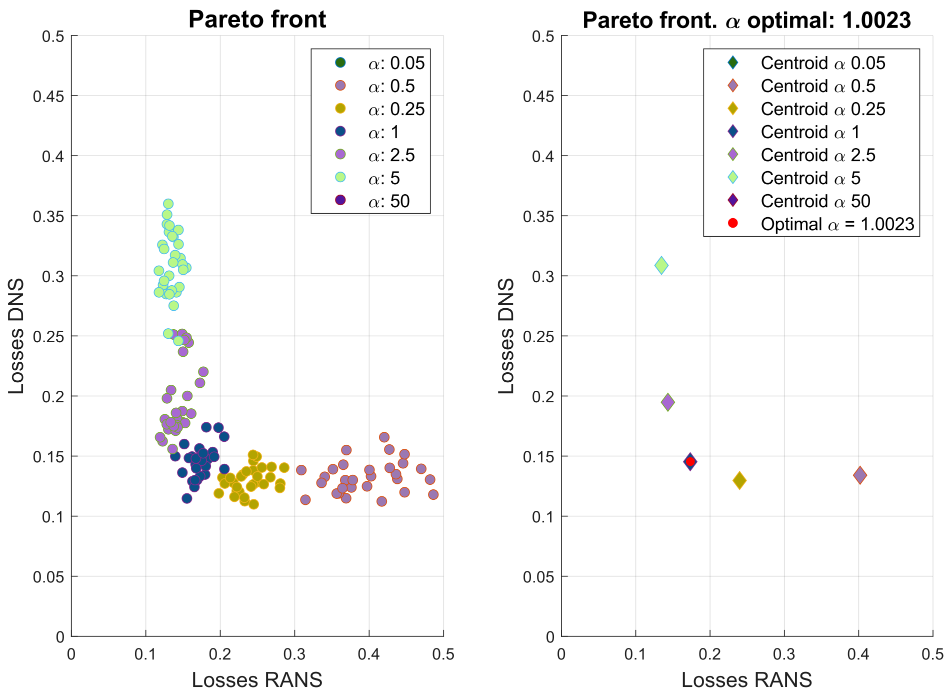


Figure 4.6: Pareto front optimization. On the left are plotted all the values $\mathcal{L}_{\mathrm{DNS},k}$ and $\mathcal{L}_{\mathrm{RANS},k}$ for each value of α_k represent by a colour. For every α_k, for statistical purposes, a cloud of points is computed as can be seen. On the right, the centroid through the 30 statistical training for each α_k has been computed. The red dot on the right figure is the closest point to the origin.

In a Pareto front optimization, according to [31], three sets of points can be defined:

- The Utopia point which is the location in the Pareto front that represents the lower bound among all possible objectives taken individually. This point can also be referred to a non-existent solution. In this case, the utopia point is the origin of the axis since is the minimum for both losses;
- The Nadir point which is the location in the Pareto front that represents the upper bound among all possible objectives taken individually. In this case, is the point in the space composed by $[\boldsymbol{\mathcal{L}}_{\mathrm{RANS}}; \boldsymbol{\mathcal{L}}_{\mathrm{DNS}}]$ with components [MAX $(\mathcal{L}_{\mathrm{RANS}})$; MAX$(\mathcal{L}_{\mathrm{DNS}})$];
- The Optimal point which is the location in the Pareto front that is the trade-off between the 2 objective variables. It can be mathematically expressed as the nearest feasible point to the utopia one. In Figure 4.6 is approximately $[0.17; 0.15]$

In Figure 4.6 is shown the result of the Pareto front optimization. As can be seen the two variables $\mathcal{L}_{\text{DNS}}$ and $\mathcal{L}_{\text{RANS}}$ are conflicting, when one decreases the other one increases. The α optimal point was computed by interpolating along the line and finding the point that is closest to the utopia point.

This type of analysis showed that the best value for α is equal to one, which means that an equal weight to the DNS and the RANS part should be done. The network used for future analysis will be the one with $\alpha = 1$.

In Figure 4.7 is represented the α value interpolation. It's clear that the closest point to the utopia point is the one shown with the red rhombus. The red circle shows this distance. In Figure 4.7 can be stated that for a vast range of α the distance from the utopia point doesn't change a lot. This behavior is due to the fact that the Pareto front obtained is sharp.

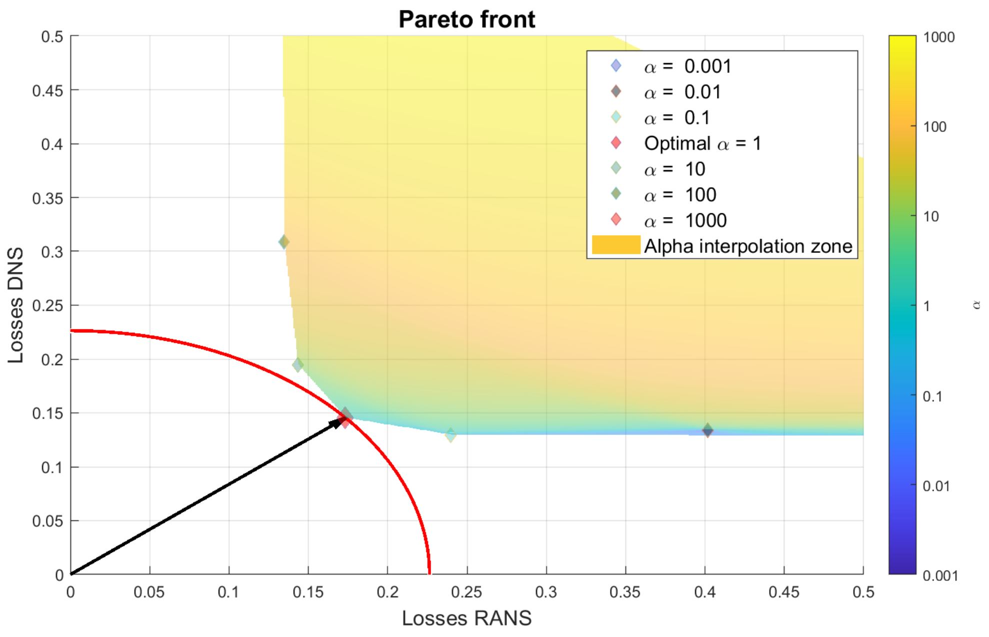


Figure 4.7: The figure shows another visualization of the Pareto front, with the α interpolation inside. This plot shows very clearly, with the red circle, that the optimal value for α is equal to 1 since it minimizes the distance from the utopia point.

Figure 4.8 shows the DNS and RANS values for the loss function varying α. In this figure can be seen clearly that by increasing the value for α the RANS losses decrease and the DNS ones have an opposite trend.

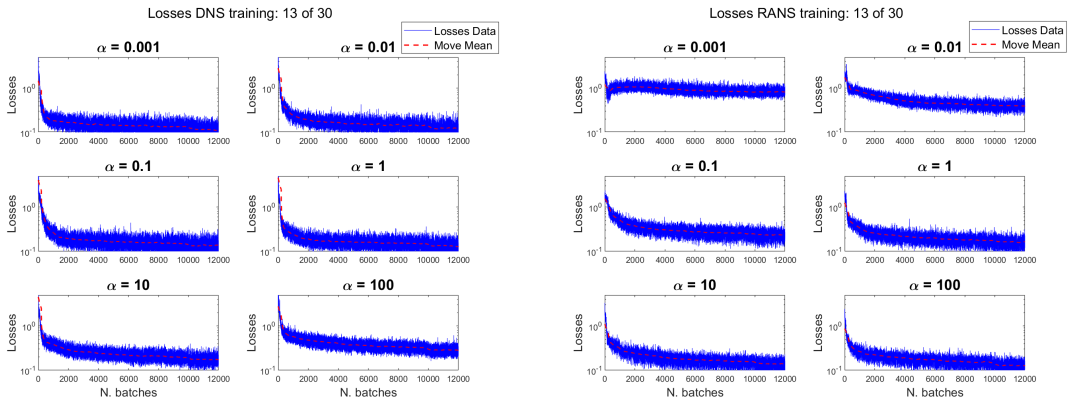


Figure 4.8: The figure on the left shows the loss trend computed with DNS inputs, the dotted red line is the move mean. The figure on the right shows the RANS losses with the increasing α value.

4.4 Pareto Front - No Thermal Part

Among the inputs introduced in Table 2.2, π_7 is the most critical one since it alters the linear dependency between the heat flux and the temperature gradient and makes the model non-linear with respect to the temperature. Hence, a new Pareto front analysis was carried out to understand the influence of this parameter on the results.

The training procedure is the same as section 4.3 performing 32 trainings for each α. Considering a new vector for α composed by:

$$\alpha = [0.05,\ 0.25,\ 0.5,\ 1,\ 2.5,\ 5,\ 50] \tag{4.4}$$

the total number of trainings performed was 224.

This new optimization was performed by removing the parameter π_7. The definition of π_7 according to Table 2.2 is the following:

$$\pi_7 = \frac{||\nabla\theta||}{\sqrt{k_\theta}}\frac{k^{1.5}}{\epsilon}$$

This has been done by placing the π_7 input at zero because it is the only parameter that contains ∇T. By looking at how the Pareto changes it is possible to evaluate the importance of this parameter on the DNS and RANS losses.

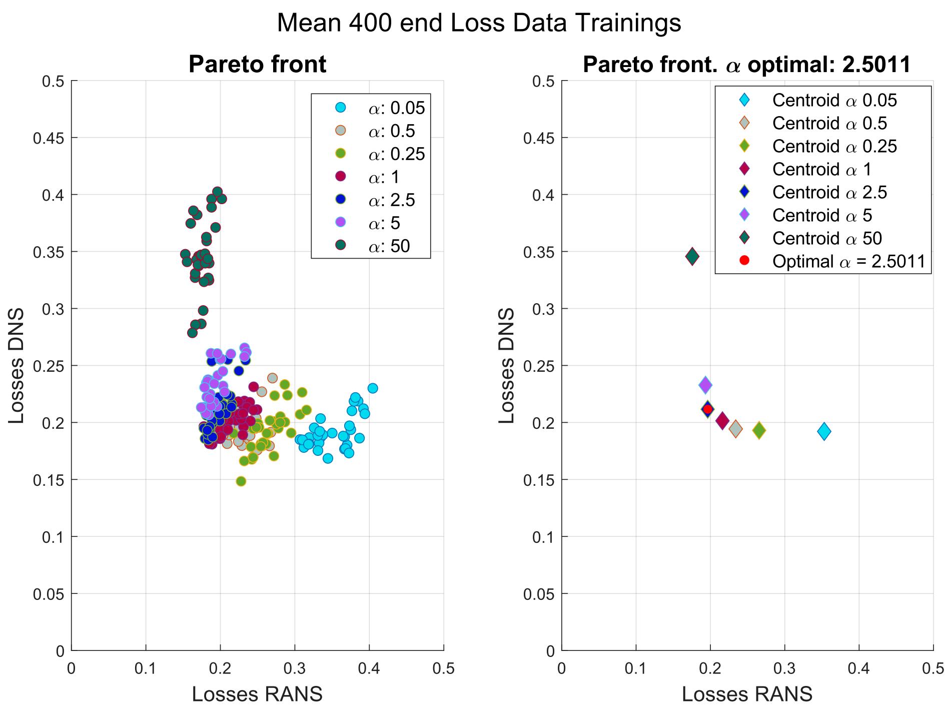


Figure 4.9: On the left-hand side is represented the mean of the last 400 DNS and RANS losses values. On the right is shown the computed centroid of the previous values. It can be seen also that the optimal point was for $\alpha = 2.5$.

In the new Pareto front, shown in Figure 4.9, is evident that the new optimal point is for an $\alpha = 2.5$. As the loss function is the following:

$$\mathcal{L} = \mathcal{L}_{\text{DNS}} + \alpha \; \mathcal{L}_{\text{RANS}}$$

can be stated that if the optimal α value increased, the DNS part is more sensitive to the thermal part. This statement can also be explained as a greater increase in the DNS losses, with respect to the RANS ones. Hence, the network to compare both inputs should increase the RANS losses by this α value.

The fact that the DNS inputs are more sensitive to the thermal part is something that can be seen later in chapter 6. The main reason for this behavior is due to the fact that all the DNS inputs are more accurate and the network will trust them much more than the RANS one.

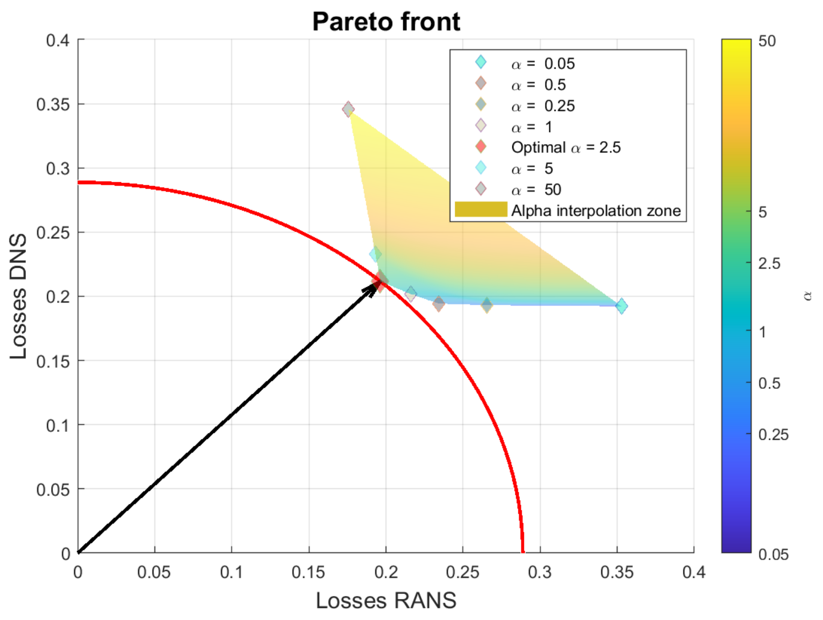


Figure 4.10: In this figure the interpolation zone of the α value is shown for the case without the thermal part. Clearly, $\alpha = 2.5$ is the nearest point to the utopia one.

4.5 Multi-objective optimization conclusions

Summarizing in this chapter the network loss function was firstly rewritten and secondly, the right trade-off between losses with DNS and RANS input was found. It was observed that the best α value that can minimize the losses obtained with DNS and RANS inputs, is equal to one. This means that an equal contribution should be attributed to both input categories. One notable result is that the network is able to behave correctly with both inputs, this is due to the fact that it is able to create connections to simulate a classifier and subsequently predict the correct output by giving weight to different inputs. In the meantime, it was also analyzed how the trainings and the Pareto front changed by removing the parameter π_7 from the inputs.

The next chapter will show the *a priori* validation (section 5.1) which consists in computing the heat flux in a database not used in the training phase and the *a posteriori* validation (section 5.2) which consist in the validation of the results into the CFD solver which is OpenFoam.

Chapter 5

Model Validation

This chapter presents and discusses the results obtained with the previous [1] and the new network architecture with DNS and $k - \epsilon$ RANS inputs. The *a priori* validation consists in computing the turbulent heat flux with inputs that are excluded from the training set. The *a posteriori* validation consists of integrating the model into a CFD solver like OpenFoam.

This chapter will be subdivided into two sections: the first one will deal with the *a priori* validation of the new model obtained in section 4.3 with $\alpha = 1$, and the second one will deal with the *a posteriori* validation again with the new model.

Recalling the formulation of the loss function as follows:

$$\mathcal{L} = \mathcal{L}_{\text{DNS}} + \alpha \, \mathcal{L}_{\text{RANS}}$$

it is clear that the previous model that was trained with only DNS data corresponds to say $\alpha = 0$ and the new model that was trained with both DNS and RANS data corresponds to $\alpha = 1$.

5.1 *A Priori* Validation $(\alpha = 1)$ model

To carry out the *a priori* validation, some simulations for the channel flow 1D, and some random points in the backward facing step 2D simulation were excluded from the training set to test the model's accuracy with new data. It's important to point out that for this validation the input set (Table 2.2) is composed as follows:

- DNS inputs: both the momentum (U, k, ϵ and ν_t) and thermal (θ, k_θ and ϵ_θ) came from a high fidelity (DNS) simulation;
- RANS inputs: the momentum part (U, k, ϵ and ν_t) came from a RANS simulation with the $k - \epsilon$ turbulence model and the thermal variables came from the corresponding DNS simulation with the same flow conditions (Re, Pr).

This section will be subdivided into 4 subsections: 2 for the validation of the channel flow with DNS and RANS inputs and 2 for the validation of the backward facing step with DNS and RANS inputs. It's important to point out that good accuracy in all cases must be achieved since the model was trained with $\alpha = 1$.

5.1.1 Channel Flow 1D with DNS inputs

In general, in a DNS simulation that aims to solve the three fluid dynamics equations (continuity, momentum, and thermal) it would not make sense to include a turbulence model to avoid one of the three equations. This validation, but also the training, in the DNS case study was therefore carried out in order to couple the neural network (closure for the thermal part) to a very precise momentum closure model such as the EBRSM (Elliptic-Blending Reynolds-Stress Model)

The results achieved in this case are shown in Figure 5.1. Where for three different Reynolds numbers and two different Prandtl numbers the turbulent heat fluxes were plotted. It's clear that the artificial neural network is able to fit the DNS heat flux value for both $Pr = 0.01$ and $Pr = 0.025$.

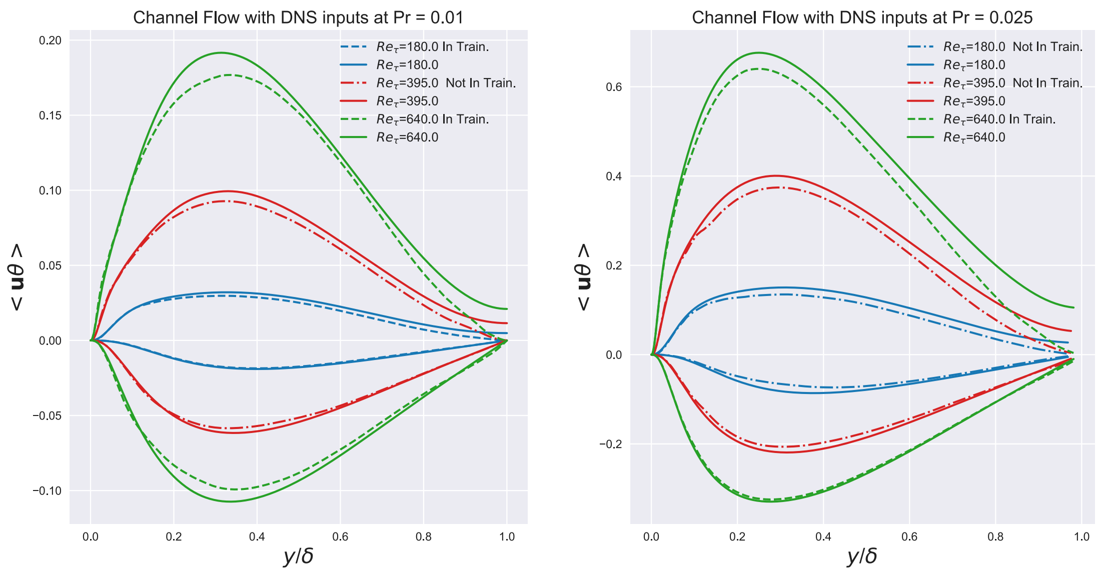
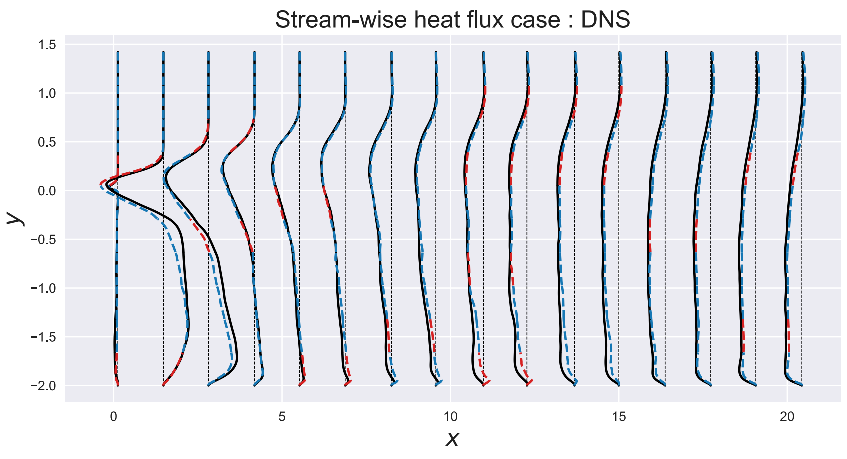


Figure 5.1: DNS inputs and $\alpha = 1$ network. On the left are shown the three Re_τ numbers for $Pr = 0.01$ and on the right with $Pr = 0.025$. In both figures, the solid line is the reference DNS heat flux, with the dotted line is represented a simulation that was in the training set and with the dash-dotted line is represented a simulation that was not in the training set. On the top of each figure is plotted the stream-wise heat flux ($\overline{u\theta}$) and on the bottom the wall-normal ($\overline{v\theta}$).

5.1.2 Backward Facing Step 2D with DNS inputs

In the backward facing step test case, random samples were taken as training inputs and validation. The test case geometry is the same of Figure 3.4 and the validation was made with $Re_b = 3200$ and $Pr = 0.01$. The Reynolds number is computed in Equation 3.3. Figure 5.2 shows respectively the stream-wise and the wall-normal turbulent heat flux.



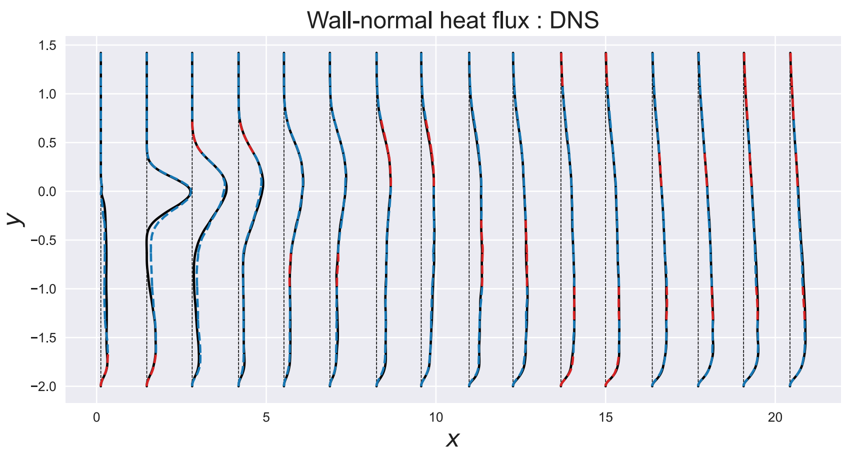


Figure 5.2: Stream-wise and wall-normal turbulent heat flux with DNS inputs and $\alpha = 1$ network in the BFS configuration with $Re_b = 3200$ and $Pr = 0.01$. The solid black line is the DNS reference heat flux. The blue dotted line is the ANN heat flux computed with inputs that are in the training set and the red dotted line is with inputs excluded from the training set.

The predicted value for the stream-wise heat flux shows small inaccuracies in the lower wall zone. This behavior is due to the limitation of the algebraic mathematical structure of the model which is inherently local and makes it very difficult to follow the evolution of the thermal field [1]. In particular, in this zone, the data-driven approach overestimates the extension of the thermal separation region.

5.1.3 Channel Flow 1D with RANS inputs

In this section are shown the results achieved for the turbulent heat flux starting from RANS inputs. The inputs were computed with a momentum model that applies the Boussinesq hypothesis, the $k - \epsilon$ in this case. Only inputs related to the momentum have experienced an approximation due to the momentum turbulence model. The thermal part has not been changed. The results made with these inputs are shown in Figure 5.3

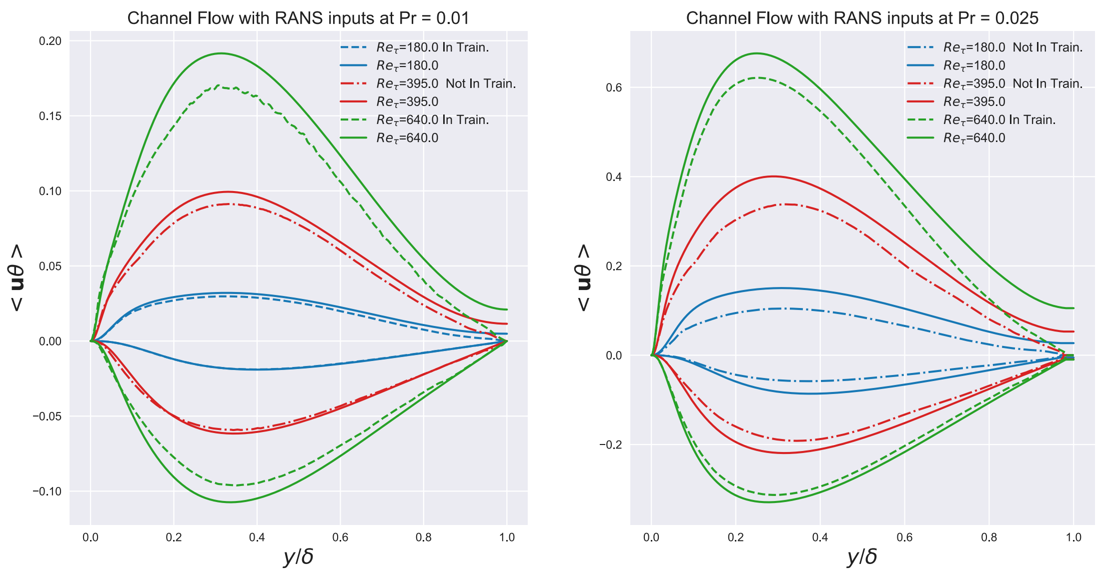


Figure 5.3: RANS inputs and $\alpha = 1$ network. On the left are shown the three Re_τ numbers for $Pr = 0.01$ and on the right with $Pr = 0.025$. In both figures, the solid line is the reference DNS heat flux, with the dotted line is represented a simulation that was in the training set, and with the dash-dotted line is represented a simulation that was not in the training set. On the top of each figure is plotted the stream-wise heat flux ($\overline{u\theta}$) and on the bottom the wall-normal ($\overline{v\theta}$).

From Figure 5.3 it's clear that the artificial neural network predictions for the turbulent heat flux (dotted, dashed-dotted) fit very well with the true DNS heat flux (solid lines). In Figure 5.4 can be seen the differences between the previous model [1] with $\alpha = 0$ and the new trained one with $\alpha = 1$.

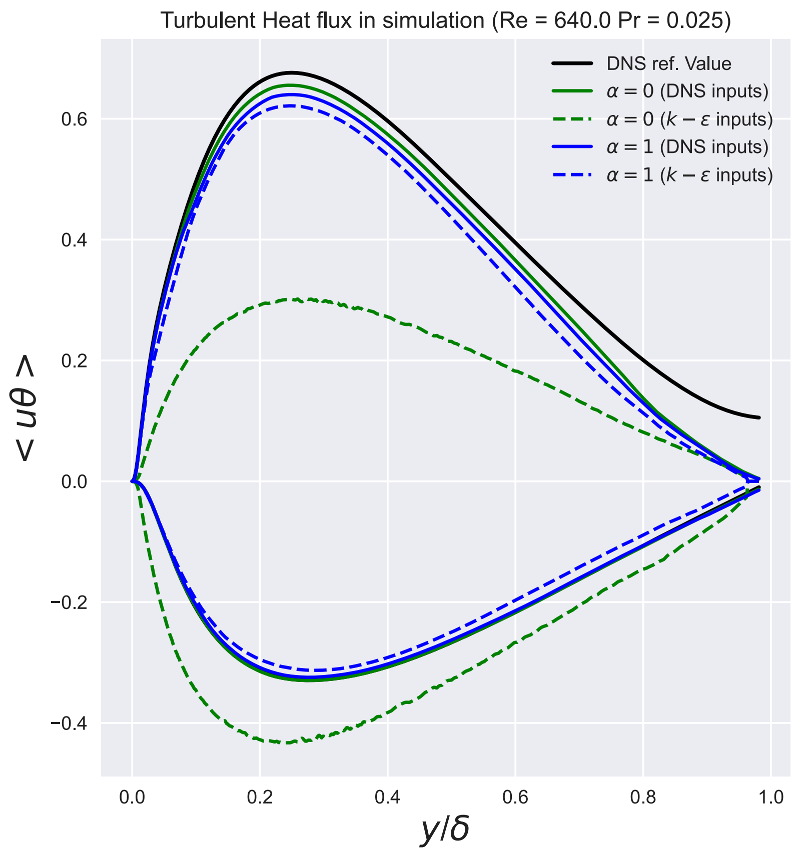


Figure 5.4: The solid black line is the true DNS turbulent heat flux, with the green lines is represented the previous network ($\alpha = 0$) behavior (solid line - DNS inputs and dotted line - RANS inputs) and with the blue line the new network ($\alpha = 1$) behavior (solid line - DNS inputs and dotted line - RANS inputs). On the top of each figure is plotted the stream-wise heat flux ($\overline{u\theta}$) and on the bottom the wall-normal ($\overline{v\theta}$).

What can be seen from Figure 5.4 is that the new trained network (blue lines) manages to be faithful to the turbulent heat flux behavior produced by the DNS simulation (black lines) with both, DNS (solid blue lines) and $k - \epsilon$ (RANS) inputs (dashed blue lines). In contrast to this, the prior network (green lines) behaves very poorly in the presence of $k - \epsilon$ (RANS) input (dashed green lines) but better with DNS inputs (solid green lines).

5.1.4 Backward Facing Step 2D with RANS inputs

The validation with RANS inputs was made also in the backward facing step test case. The geometry related to the backward facing step can be observed in Figure 3.4 and the analysis was made at $Re_b = 3200$ and $Pr = 0.01$.

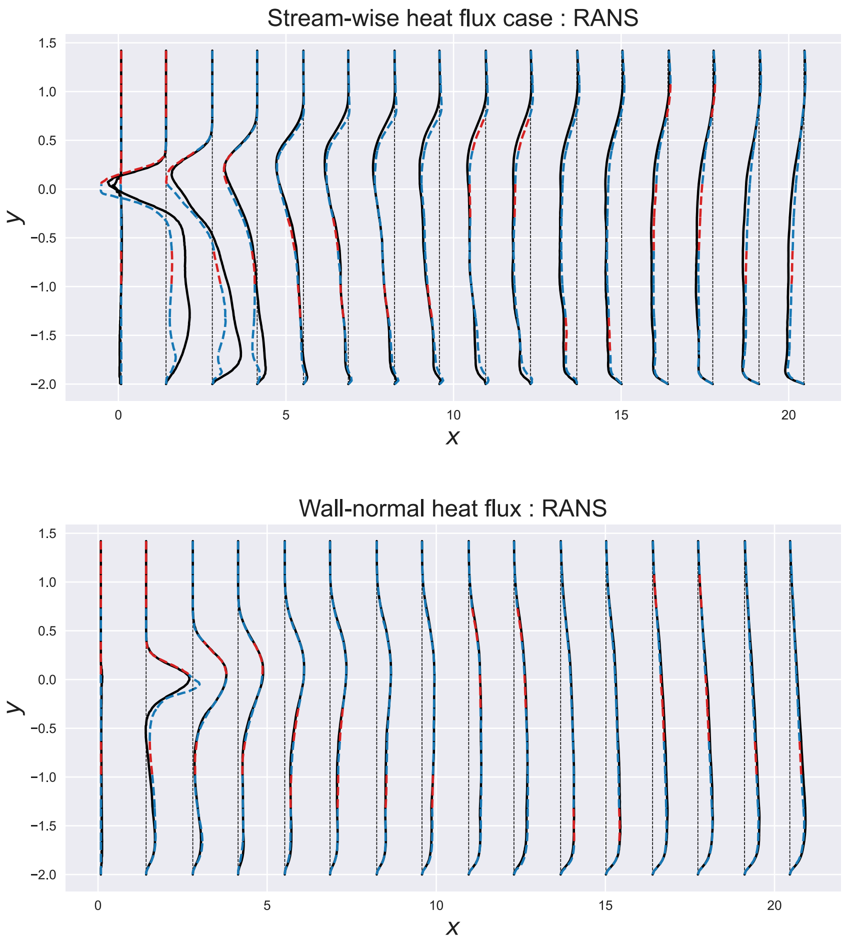


Figure 5.5: Stream-wise and wall-normal turbulent heat flux with DNS inputs and $\alpha = 1$ network in the BFS configuration with $Re_b = 3200$ and $Pr = 0.01$. The solid black line is the DNS reference heat flux. The blue dotted line is the ANN heat flux computed with inputs that are in the training set and the red dotted line is with inputs excluded from the training set.

A similar behavior of Figure 5.2 is observed for the regions close to the step. In this case with RANS ($k - \epsilon$) inputs, the performance is worse than with high-fidelity inputs. In particular, the artificial neural network overestimates the thermal separation zone more than with DNS inputs. As told before in subsection 5.1.2 this is a problem due to the mathematical formulation of the network.

5.1.5 *A Priori* validation conclusion

In conclusion, can be stated the following:

- A very good accuracy for the stream-wise and the wall-normal heat flux was obtained in channel flow with both DNS and RANS inputs;
- Good results were obtained in the backward facing step with both sets of inputs. A lower accuracy was observed in the area near the wall and the step. This issue is due to the mathematical structure of the model which is predominantly local and makes it very difficult to follow the evolution of the thermal field after the step;
- The fact that the model manages to work well with both inputs proves that the neural network constructs a kind of internal classifier that leads to different model behaviors in presence of different inputs. Indeed, it turns out to be important that it gives good results with both weak ($k - \epsilon$) and more accurate (EBRSM) momentum turbulence models;
- The fact that the results of this validation are good does not mean that the model performs well coupled with a CFD solver. This is due to the fact that in the *a priori* validation the thermal part for RANS inputs was copied from high-fidelity simulations. Therefore, it is of paramount importance to proceed with the *a posteriori* validation.

5.2 *A Posteriori* Validation

Having obtained good results through *a priori* validation, it is necessary to proceed with the implementation and the validation of the artificial neural network directly within a CFD solver. OpenFoam 8 has been chosen as the CFD software. A heat flux problem, in OpenFoam 8, can be solved with the following solvers: [32]:

- buoyantBoussinesqSimpleFoam: solver steady state for buoyant and turbulent flows in incompressible fluids;
- buoyantBoussinesqPimpleFoam: solver transient state for buoyant and turbulent flows in incompressible fluids;

- buoyantSimpleFoam: solver steady state for buoyant and turbulent flows in compressible fluids;
- buoyantPimpleFoam: solver transient state for buoyant and turbulent flows in compressible fluids;

Referring more specifically to the solver "buoyantSimpleFoam" it solves the steady state Navier Stokes equations that are shown in Equation 5.1.

$$\begin{cases} \nabla \cdot (\rho \mathbf{u}) = 0 \\ \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = -\nabla p + \rho \mathbf{g} + \nabla \cdot (2\mu_{\text{eff}} S(\mathbf{u})) - \nabla \left(\frac{2}{3}\mu_{\text{eff}} \left(\nabla \cdot u\right)\right) \\ \nabla \cdot (\rho \mathbf{u} h) + \nabla \cdot (\rho \mathbf{u} K) = \nabla \cdot (\alpha_{\text{eff}} \nabla h) + \rho \mathbf{u}\mathbf{g} \end{cases} \tag{5.1}$$

where $\boldsymbol{u}$ is the velocity field; ρ is the density field; p the static pressure; $\boldsymbol{g}$ the gravitational acceleration; μ_{eff} is the sum of the turbulent and molecular viscosity; $S(\boldsymbol{u})$ is the rate of strain tensor (Equation 2.15); $K = {}^{|u^2|}/_2$ is the kinetic energy; h is the entalpy per unit mass $h = e + {}^p/_\rho$ and e is the energy per unit mass. α_{eff} is the thermal diffusivity:

$$\alpha_{\text{eff}} = \frac{\nu_t}{Pr_t} + \frac{k}{\rho C_p} = \alpha_t + \alpha \tag{5.2}$$

If the Reynolds analogy is applied, Pr_t would be a constant value and the problem would be closed. Due to the nature of the problem, the Reynolds analogy cannot be applied (analyzed in subsection 2.1.2) and the closure goes directly to the turbulent heat flux with Equation 2.16 applying an artificial neural network.

$$\underbrace{\overline{\boldsymbol{u'\theta'}} \neq -\alpha_t \frac{\partial \theta}{\partial x_i}}_{\text{Reynolds Analogy}} \Longrightarrow \underbrace{\overline{\boldsymbol{u'\theta'}} = -\underbrace{\mathcal{M}(\boldsymbol{X}, \mathcal{W})}_{=\boldsymbol{D}} \nabla \theta}_{\text{Neural Network}}$$

The solver has been modified firstly in order to compute the inputs, as seen in Table 2.2, and secondly to import the neural network and run the network's forward pass calculating the turbulent heat flux.

The network has been trained using the PyTorch framework in Python and it has to work in OpenFoam which is written in C++. It is therefore necessary to find a way to allow the PyTorch network to work in OpenFoam. Implementing the PyTorch APIs in OpenFoam was observed to be a very difficult task and therefore the same methodology applied in the previous work [1] was followed. To enable the PyTorch network to work in OpenFoam, the network was first converted to a standard format for neural networks (ONNX) and then into Tensorflow. Once

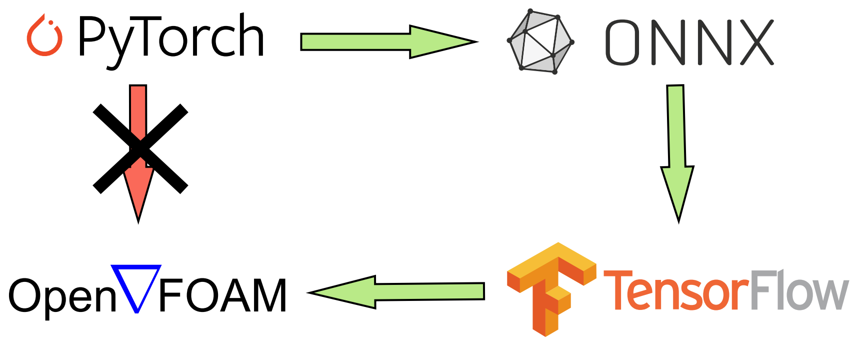


Figure 5.6: PyTorch network implementation in OpenFoam. The network is firstly converted to ONNX (Open Neural Network Exchange) then into Tensorflow (a machine learning library) and in the end the Tensorflow APIs implementation in OpenFoam.

the network was saved in a format readable through the Tensorflow library the Tensorflow APIs were implemented in OpenFoam. Figure 5.6 shows the workflow for importing the network into OpenFoam.
To better explain the implementation process, a GitHub repository has been created. The README file showing the detailed procedure can be found in Appendix C. A summary of what has been done for this validation is

- The use of the OpenFoam solver "buoyantSimpleFoam";
- The network conversion from PyTorch to Tensorflow (section C.1);
- The creation of a new thermal turbulence model in OpenFoam;
- The implementation of the TensorFlow APIs in the new model (section C.3);
- The validation of the model with two different test cases: the channel flow (subsection 5.2.1) and the Impinging Jet (subsection 5.2.2).

It is important to note that all the neural network inputs, in this case, are from a RANS simulation with a momentum turbulence model that applies Boussinesq's hypothesis such as the standard $k - \epsilon$.

5.2.1 Channel Flow *a posteriori* validation

The channel flow is the standard test case since it was used both for training the neural network and also in the *a priori* validation.

The simulation setup is the same as the one that has been used for the generation of the RANS inputs in section 3.1. The simulation conditions were the same as

the DNS test case of the Kawamura datasets [23] and the OpenFoam outputs were converted to adimensional quantities. Test case geometry was that of Figure 3.1 with the constant wall heat flux imposed. Since the equations in OpenFoam are dimensional, the half-width of the channel was chosen as $\delta = 3.025\,\mathrm{cm}$, the viscosity as $\mu = 1.74 \times 10^{-7}\,\mathrm{Pa\,s}$, the density as $\rho = 1\,\mathrm{kg\,m^{-3}}$ and the heat capacity as $C_p = 1\,\mathrm{J\,kg^{-1}\,K^{-1}}$.
The simulation conditions are the following:

- $Re_\tau = 640$ defined in section 3.1;
- $Pr = 0.025$;
- The velocity initial condition is the bulk velocity v_b computed reversing Equation 5.3;
- The velocity boundary conditions are: periodic in the stream-wise direction, empty in the span-wise direction (no equation solved in this direction), and a no-slip boundary condition is imposed at the wall;
- The temperature boundary condition imposes a constant heat flux on the wall. By deriving the dimensionless equation, an averaged turbulent heat flux is added;
- The mesh is composed by 10 000 cells with 50 cells in the stream-wise direction
- Launder-Sharma $k - \epsilon$ momentum as turbulence model which is described in subsection 2.1.1;
- Neural network model for the turbulent heat flux with $\alpha = 1$ which is the weight described in chapter 4.

According to Kawamura [23] for $Re_\tau = 640$ the bulk Reynolds number is $Re_b = 24\,428$ which is defined as follows:

$$Re_b = \frac{2\delta v_b}{\nu} \tag{5.3}$$

which lead to compute the bulk velocity $v_b = 7.03\,\mathrm{cm\,s^{-1}}$. For the temperature a constant heat flux is imposed:

$$n_j \frac{\partial T_1}{\partial x_j} = -\frac{T_d}{\delta}\mathrm{Pr}\mathrm{Re}_\tau \tag{5.4}$$

where n_j is the wall normal vector, $T_d = 1$ is imposed and T_1 is modified temperature defined as follows:

$$T_1 = T_0 - T + \frac{q}{\rho C_p \delta v_b} x_1 = 529\,\mathrm{K\,m^{-1}}$$

Firstly the solver worked with the Launder-Sharma $k-\epsilon$ for the momentum part coupled with the Reynolds analogy for the thermal closure in order to reach the convergence in the momentum field. The results for $U+$, $k+$ and $\epsilon+$ are represented in Figure 5.7. The temperature plot is shown in Figure 5.8.

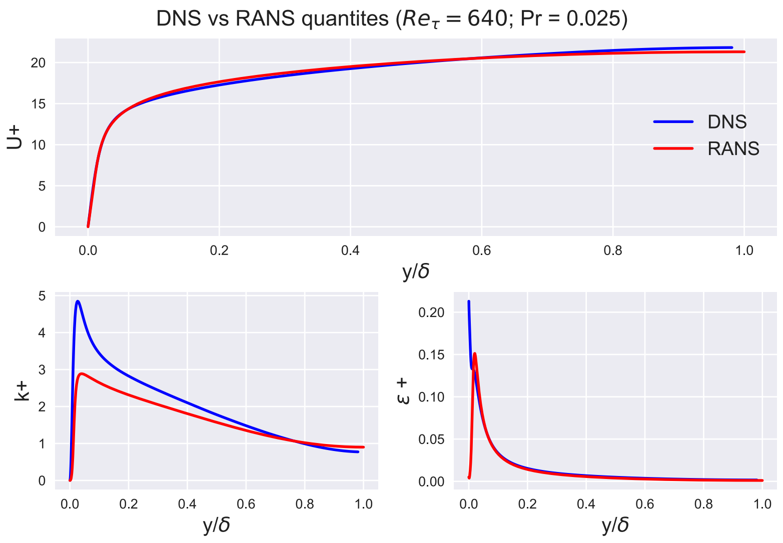


Figure 5.7: $U+$, $k+$ and $\epsilon+$ behaviour after convergence reached with Launder-Sharma $k-\epsilon$ and Reynolds analogy.

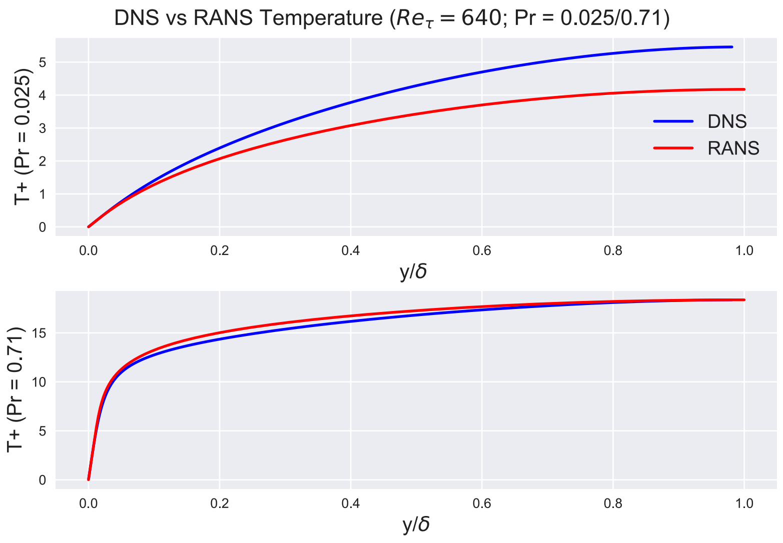


Figure 5.8: Temperature profile with Reynolds analogy. Can be seen clearly that the Reynolds analogy cannot be applied with low Prandtl numbers.

From Figure 5.7 can be stated that the velocity profile, k, and ϵ reached the convergence. In particular, the velocity profile is very similar to the DNS value, the turbulent kinetic energy and the turbulent dissipation rate are quite different in the near wall region. Instead, Figure 5.8 clearly shows that the temperature profile computed with the Reynolds analogy only matches the DNS temperature profile if the simulation has a sufficiently high Prandtl number ($Pr = 0.71$). If the Prandtl number is low, it's clear that the two curves don't match.

The next step was to change the turbulence model from the Reynolds analogy to the artificial neural network model. In this case, the turbulent heat flux is predicted with the neural network using the input of the current state, and the subsequent updating of the parameters k_θ and ϵ_θ after solving their equation with the predicted turbulent heat flux. The residuals in the k_θ and ϵ_θ equations are plotted in Figure 5.9.

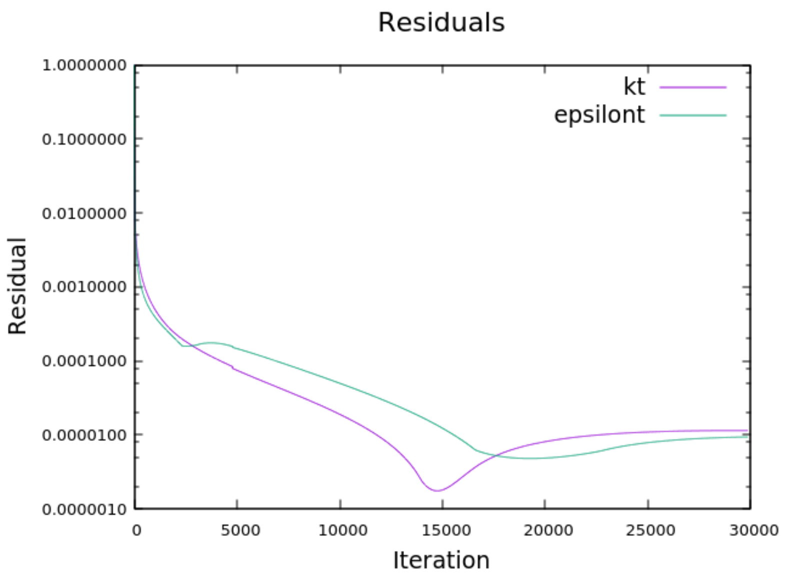


Figure 5.9: Thermal variance (k_θ) and thermal dissipation rate (ϵ_θ) residuals with the neural network implemented as thermal closure model.

After reaching the convergence for k_θ and ϵ_θ, the turbulent heat flux can be plotted to validate the neural network model. The turbulent heat flux is shown in a Figure 5.10. The turbulent heat flux is very similar to the DNS value this means that the model works in a good way to achieve the thermal closure for this test case.

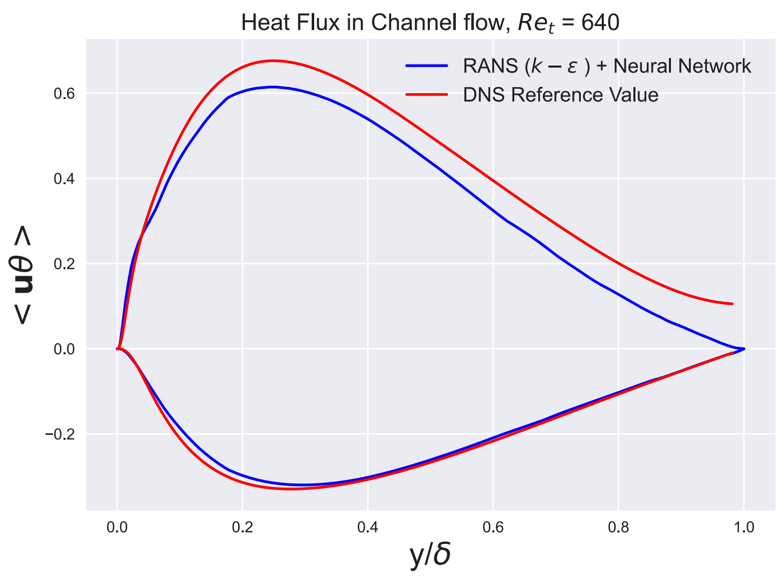


Figure 5.10: On the top the stream-wise heat flux $(\overline{u\theta})$ and on the bottom the wall-normal heat flux $(\overline{v\theta})$ for $Re_\tau = 640$ and $Pr = 0.025$. With the red line is shown the reference turbulent heat flux, with the blue line the heat flux computed with the Launder-Sharma $k - \epsilon$ momentum turbulence model and the artificial neural network.

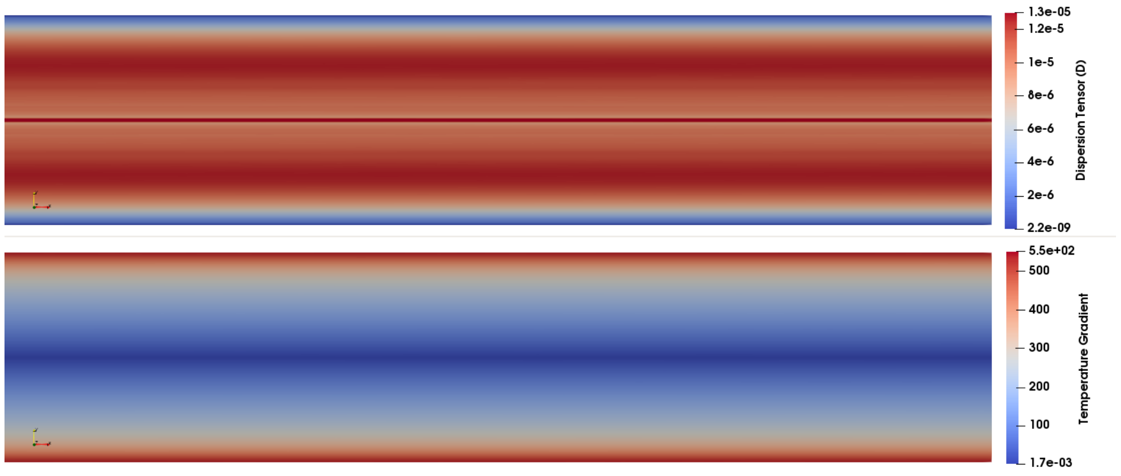


Figure 5.11: On the top is represented the dispersion tensor, on the bottom the temperature gradient. It can be noted clearly that since the temperature gradient in the center line is zero the dispersion tensor tends to be very high.

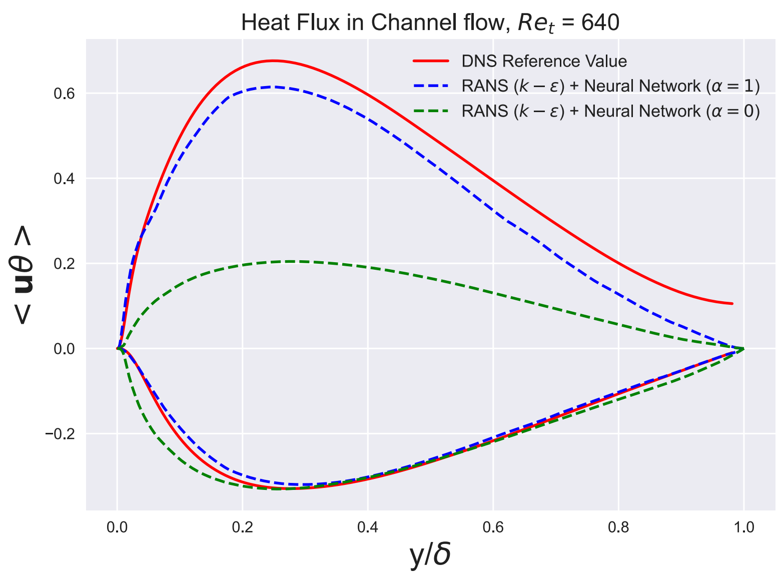


Figure 5.12: Stream-wise heat flux $(\overline{u\theta})$ and wall-normal heat flux $(\overline{v\theta})$ for $Re_\tau = 640$ and $Pr = 0.025$. With the red line is shown the reference turbulent heat flux, with the blue dotted line the heat flux computed with the $k-\epsilon$ momentum turbulence model and the artificial neural network with $(\alpha = 1)$ and with the dotted green line with $(\alpha = 0)$.

Figure 5.11 shows the dispersion tensor D at the top and the temperature gradient ∇T at the bottom, since the turbulent heat flux is computed according to Equation 2.16. The cause of the singularity value for D should be sought in the stream-wise heat flux at $y/\delta = 1$. For example, in Figure 5.10 it is clear that at $y/\delta = 1$, the predicted stream-wise heat flux, is zero but the DNS reference one is non-zero. At this point, a singularity for D should exist since the temperature gradient is vanishing.

In Figure 5.12 is shown the comparison between the turbulent heat flux computed with the new network ($\alpha = 1$, blue line) and with the previous network ($\alpha = 0$, green line). It can be seen from the above figure that the thermal turbulence model based on the old artificial neural network ($\alpha = 0$) was not able to correctly predict the value of the stream-wise heat flux. The behavior of the *a posteriori* validation of the two artificial neural networks is very similar to the *a priori* validation shown in Figure 5.4.

In conclusion, it is possible to say that the model trained in chapter 4 behaves correctly in both *a priori* and *a posteriori* validation. It is therefore necessary to proceed with the validation of the model in a more complex test case to evaluate its capabilities and applicability in more common cases.

5.2.2 Impinging Jet *a Posteriori* validation

The impinging jet is an important benchmark test case used in validating thermal turbulence models that are solving the Reynolds Averaged Navier Stokes equations. This is why it involves complex flow physics and the heat transfer mechanism. These elements are important for many practical engineering applications. In this thesis work, this test case turns out to be very important since it had not been used previously in the training set.

In an impinging jet test case, a high-speed jet flow travels straight toward a solid surface, where it impinges and spreads out with a sort of circular pattern. This behaviour creates a stagnation point and leads to strong turbulence mixing and heat transfer. An overview of the behaviour of the velocity field can be observed in Figure 5.13.

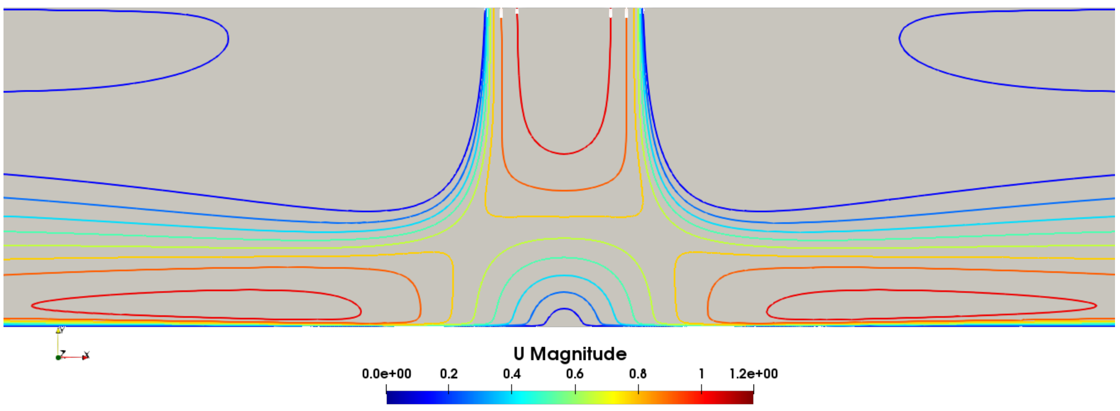


Figure 5.13: The iso-velocity lines in the impinging jet test case. Can be noted the inlet flow on the top and the impinging with the lower wall cause 2 recirculation zones.

The impinging jet test case is widely investigated in both experimental and numerical test cases with $Pr \approx 1$ as in [33]. In addition, this test case constitutes a canonical test case of interaction between a wall and a turbulent flow. On the other hand, it constitutes also an efficient way to cool a surface that is widely used in gas turbine blade cooling, electronic cooling and chemical processing. As for the last one, an impinging jet can be used as a cooling method for nuclear reactor components such as fuel rods or reactor vessel walls.

There are many configurations of an impinging jet. Most of them depend on the shape of the jet (round or planar), the ratio between the width of the jet and its diameter, and the distance between the nozzle and the impinging wall [34].

The DNS simulation used as a reference for this test case for low Prandtl number was carried out by M. Duponcheel and Y. Bartosiewicz in [34]. In the previous paper, the DNS simulations were performed at $Re = 4000$ and $Re = 5700$ based on the jet width and velocity and for $Pr = 1$, $Pr = 0.1$, and $Pr = 0.01$.

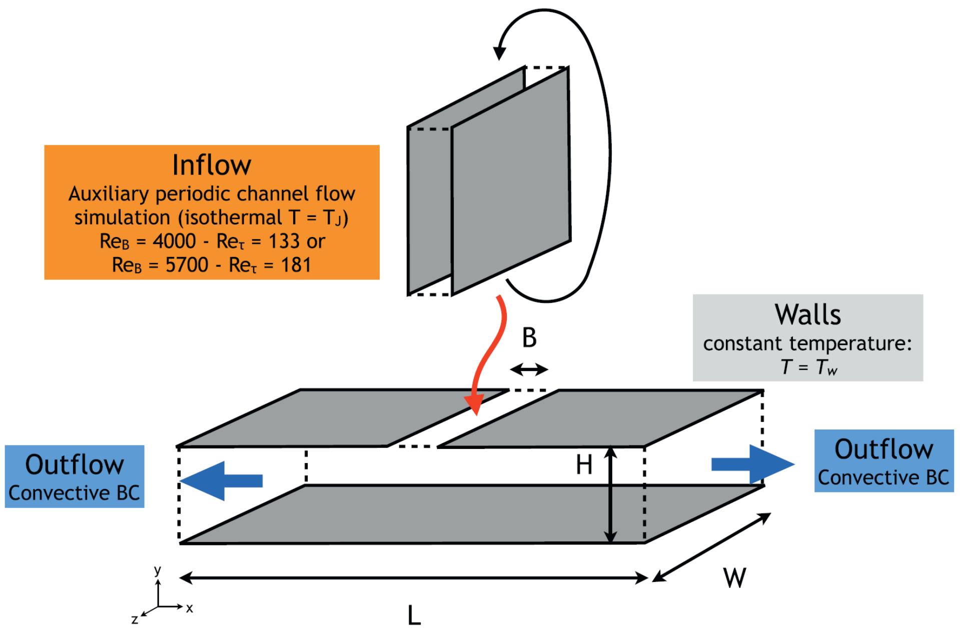

Inflow
Auxiliary periodic channel flow simulation (isothermal T = T$_J$)
Re$_B$ = 4000 - Re$_\tau$ = 133 or
Re$_B$ = 5700 - Re$_\tau$ = 181
Walls
constant temperature: $T = T_w$
B
H
Outflow
Convective BC
Outflow
Convective BC
L
W
y
x
z

Figure 5.14: Impinging jet geometry with developed turbulent flow profile [34].

The geometry is represented in Figure 5.14 and it consists in two infinite parallel flat plates where the top one is split in order to inject the flow. The distance from the two flat plates is H and the slit dimension is B. The top and the bottom walls are isothermal so $T = T_w$ and $u_i = 0$ at $y = 0$ and when $y = H$ and $|x| \geq B/2$ [34]. The setup is purely forced convection and the temperature is treated as a passive scalar. The flow can be characterized by its Reynolds number computed as follows:

$$Re = \frac{UB}{\nu} \tag{5.5}$$

where U is the mean jet velocity. This Reynolds number is also the bulk Reynolds number of the auxiliary channel flow which can be also characterized by:

$$Re_\tau = \frac{u_\tau (B/2)}{\nu} \tag{5.6}$$

where $u_\tau = \sqrt{\tau_w/\rho}$ is the friction velocity and τ_w is the mean shear stress. The turbulent inlet flow is simulated in a parallel, but separate, infinite channel flow with periodic boundary conditions as seen in subsection 5.2.1. The temperature field can be characterized by its Pr number. Another important value to characterize the impinging jet is the Aspect Ratio ($AR = H/B$). A higher aspect ratio may be desirable in order to be closer to a free impinging jet geometry but can be very computationally expensive since they require a longer domain [34]. A small AR of 2 is considered for the validation.

The RANS simulations were carried out by [35] with different momentum models (Launder-Sharma $k-\epsilon$, $k-\omega$ SST, v2-f) and thermal closure models (Reynolds analogy, Key's correlation, Manservisi and the AHFM-NRG). The RANS setup has a bottom wall length of $L = 40B$; the domain height is $H = 2B$ and the domain thickness is $W = 3.14B$. The inflow temperature is considered as 1 and the temperature of the walls is 0.

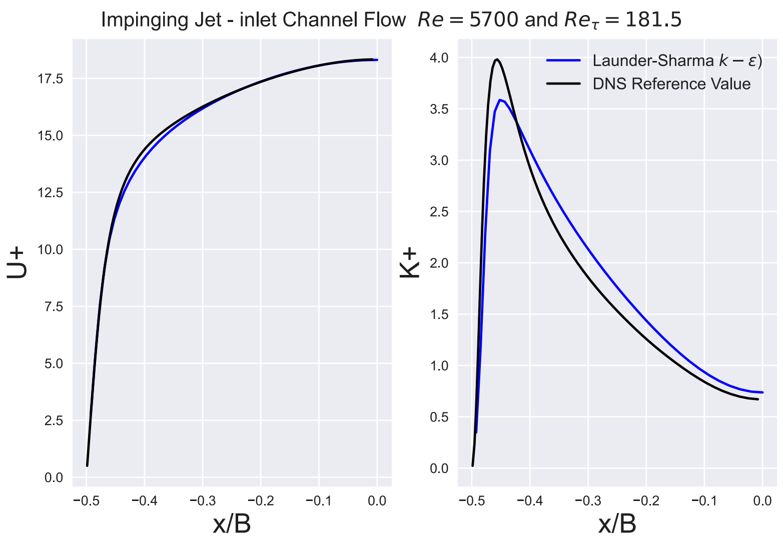
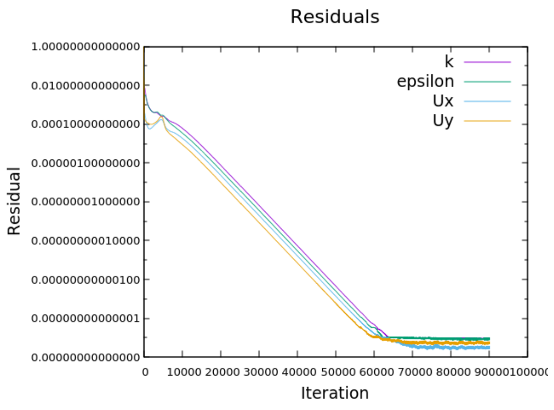


Figure 5.15: Inlet velocity profile normalized with friction velocity on the left and turbulent kinetic energy normalized with the square of the friction velocity on the right.

The following validation was done by keeping the exact same setup of [35] but changing the thermal closure model using the trained artificial neural network with $\alpha = 1$. Precisely as it has been done before in the channel flow, the simulation was made firstly with the Launder-Sharma $k-\epsilon$ momentum turbulence model coupled with the Reynolds analogy (for the thermal part) and secondly, after reaching the convergence for the momentum part, with the artificial neural network as thermal closure model. In Figure 5.15 are represented the inlet velocity and turbulent kinetic energy after the momentum convergence.



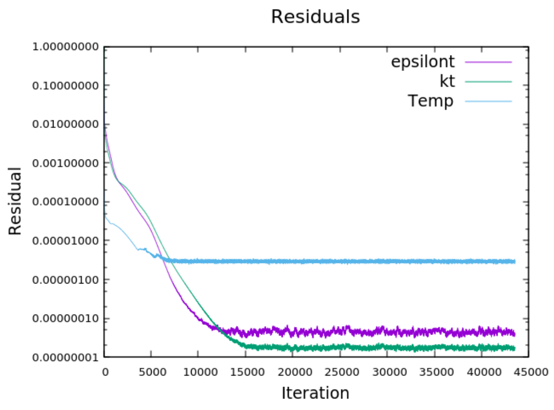


Figure 5.16: On the left are the residuals with the Reynolds analogy and on the right the residuals with the neural network thermal model.

In Figure 5.16 are represented the residuals achieved coupling the Launder-Sharma $k - \epsilon$ as momentum closure and the Reynolds analogy as thermal closure on the left. Instead, on the right are shown the residuals for the thermal closure coupling the Launder-Sharma $k - \epsilon$ that already reached the convergence and the artificial neural network. The residuals in this last case are for k_θ, ϵ_θ, and the temperature. The DNS data for this test case were provided at some x/B coordinates in according to [34]. The validation was therefore carried out at these coordinates.
The result achieved with the Launder-Sharma $k-\epsilon$ coupled with the artificial neural network were compared with the DNS reference data and with the Manservisi and Menghini thermal model (subsection 2.1.2) coupled with the Launder-Sharma $k - \epsilon$. In Figure 5.17 can be observed the wall-normal turbulent heat flux ($\overline{v'\theta}$) and in Figure 5.18 the temperature profile. In these figures, the black lines are the reference wall-normal heat flux or temperature, the purple lines are the ones computed with the Launder-Sharma $k - \epsilon$ coupled with the Manservisi thermal model and, the green lines are the ones computed with the Launder-Sharma $k - \epsilon$ coupled with the artificial neural network.
In Figure 5.17 can be seen very clearly that the turbulent heat flux computed with the neural network model behaves better than the Manservisi model near the slit (x/B = 1). Away from the slit, the two models (Manservisi and the neural network) behave more or less in the same way. The neural network still remains closer to the DNS reference value in most coordinates.
In Figure 5.18 are represented the temperature profiles for $Pr = 0.01$. The temperature close to the impingement is quite well-predicted. In the proximity of the recirculation zone (upper wall), the temperature profile is poorly predicted. This behavior also occurs downstream as the temperature seems over-predicted.

This behavior is due to the momentum turbulence model (Launder-Sharma $k - \epsilon$) that poorly predicts the recirculation zone.

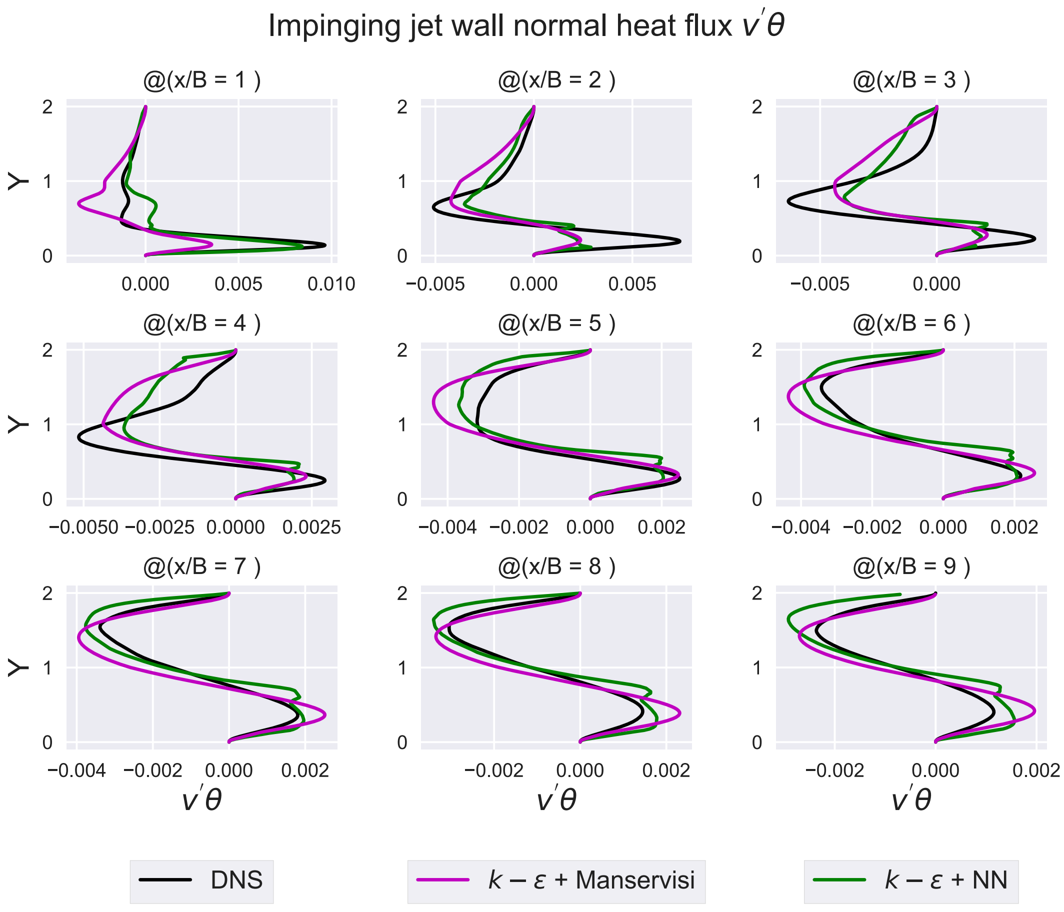
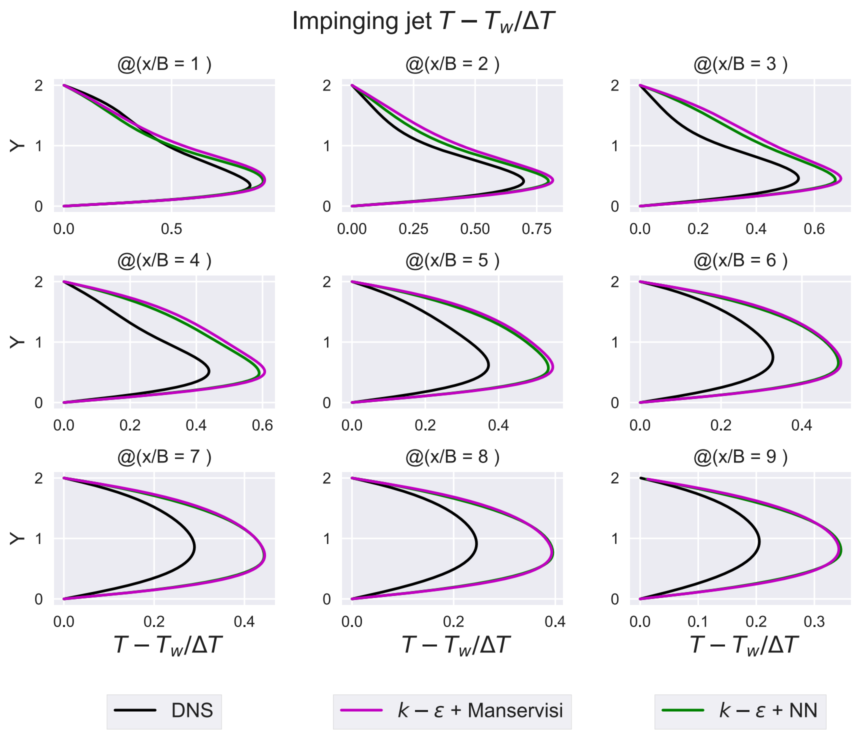


Figure 5.17: Wall normal heat flux at different x/B coordinates. The black line shows the reference heat flux (DNS), the green line computed with the Launder-Sharma $k - \epsilon$ coupled with the neural network, and the purple line computed with the Launder-Sharma $k - \epsilon$ coupled with the Manservisi model.



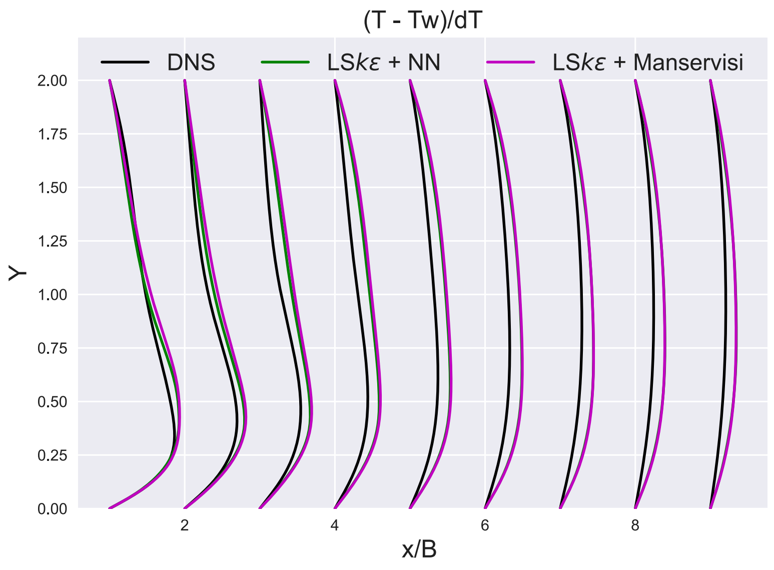
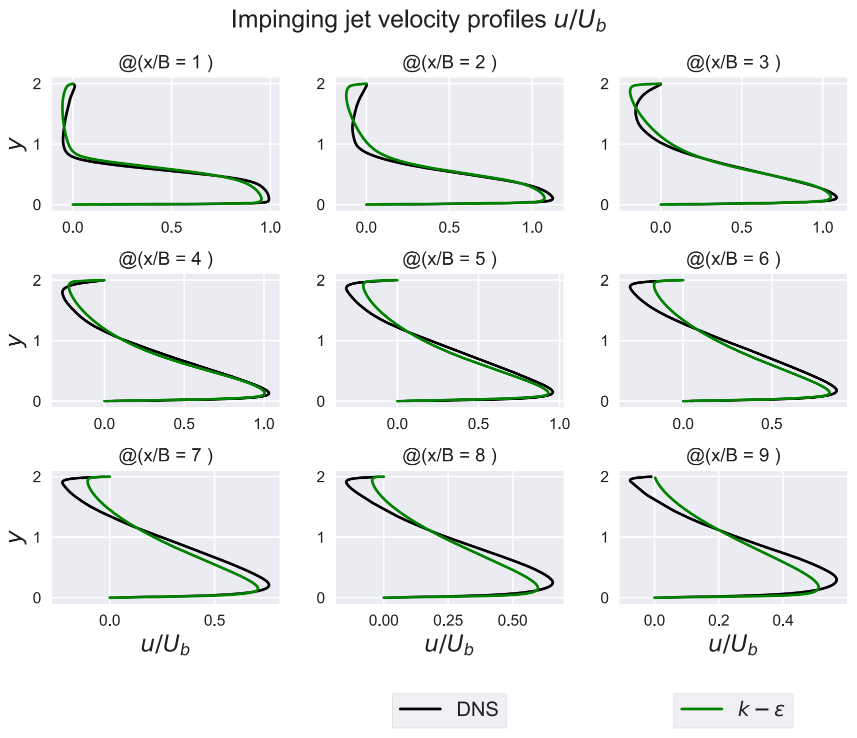


Figure 5.18: Temperature profiles at different x/B coordinates. The black line shows the reference temperature (DNS), the green line the one computed with the Launder-Sharma $k - \epsilon$ coupled with the neural network, and the purple line the one computed with the Launder-Sharma $k - \epsilon$ coupled with the Manservisi model.

The temperature results seen in Figure 5.18 are highly dependent on the momentum part due to the temperature transport with the velocity. In particular, the test case mentioned is a specific case of fluid cooling. The higher the speed, the more cooling there is. The velocity profiles are represented in Figure 5.19 in which can be observed that the velocity is under-predicted by the Launder-Sharma $k - \epsilon$ model on the bottom wall. This behavior slows down the fluid cooling in resulting a higher temperature.



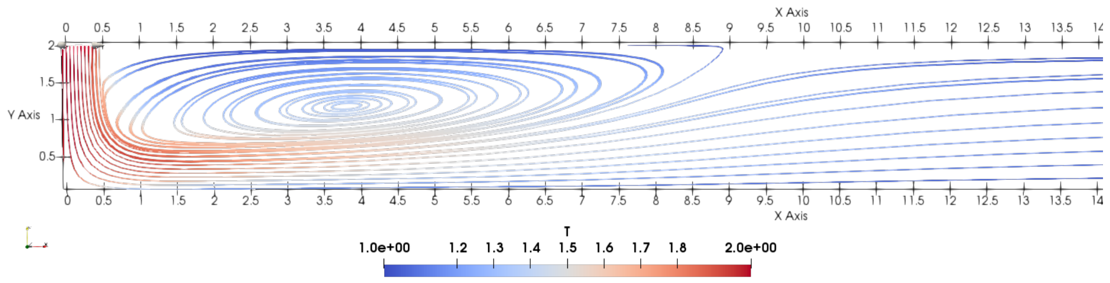


Figure 5.19: Velocity profiles, on top, in which the green line is the velocity computed with the Launder-Sharma $k - \epsilon$ momentum turbulence model and the black line is the true DNS velocity. On the bottom, the streamlines that are colored by the temperature show the recirculation zone.

5.2.3 *A posteriori* validation conclusions

In conclusion, what was carried out in this section was to implement the neural network in a CFD solver and visualize the results. Specifically, it was shown how to implement the network, which equations are solved, and the results were shown in two different geometries, one of them not in the training. The network used was the one trained with both DNS and RANS data ($\alpha = 1$) and was compared with the previous one trained only with DNS data ($\alpha = 0$). This validation made it possible to verify that indeed, the artifice that had been made to provide a value to the RANS thermal part in the *a priori* validation, did not carry an overdependence on the latter. After this work and having analyzed the results of this validation it can be deduced:

- Very good results regarding channel flow 1D (geometry used for training) for $Re = 640$ and $Pr = 0.025$ values. The heat flux value remains very close to the DNS value even using a moment closure model that leaves inaccuracies in the Reynolds tensor and in the turbulent kinetic energy value;
- The problem of a point of singularity regarding the tensor D at the center of the channel is observed. This problem was observed in Figure 5.11. However, the problem also recurs in the impinging jet and can only be solved by a different input formulation;
- Large improvements in channel flow turbulent heat flux compared with the model trained before even with inaccurate moment closure models such as the Launder-Sharma $k - \epsilon$ (Figure 5.12);
- The value of heat flux in the impinging jet turns out to be better if the Launder-Sharma $k - \epsilon$ model is coupled with the neural network developed in this thesis work, which compared then the classical Manservisi thermal model;
- The model developed always remains highly dependent on the model used for momentum closure. Inaccuracies are observed with the temperature profile due mainly to this.

Chapter 6

Interpretability Analysis

One of the main problems concerning artificial intelligence is the fact that all these algorithms are represented by a black box in the sense that it is difficult to fully understand what is inside it. A black box algorithm allows the user to see inputs and produce outputs but does not provide a detailed view either of the learning process or openly share how and why the model produced that particular conclusion. Deep learning, the technique that was used to train the model in this thesis work, in particular, uses a large number of hidden layers that are composed of a multiplicity of weights and biases leading to a complex network structure and thus difficulty in understanding. The relationship between input and output is thus obscured by this complex structure that had to be used to capture the physical phenomenon [36].

It is difficult to define interpretability, but a definition given by Miller in [37] is: "Interpretability is the degree to which a human can understand the cause of a decision".

The purpose of science is to acquire knowledge. Many problems are solved with large datasets and black-box algorithms. In this way, the model becomes the source of knowledge as opposed to data. An interpretability analysis allows the extraction of this knowledge from the model as well.

Interpretability analysis is therefore of paramount importance for multiple reasons:

- Interpretability analysis makes it possible to build trustworthy and reliable models. It allows an explanation to be provided to the user of the Machine learning model. This helps to trust the model and justify the actions that need to be taken;

- Identifying bias in models. Machine learning models often collect biases in the training phase. This can lead the model to discriminate certain data that do not have that bias. Therefore, it turns out that this type of analysis is essential for detecting biases;

- Compliance and regulatory requirements. Often industries such as finance and health care are regulated and therefore need an explanation of the model to make it more transparent;
- Improved decision-making. It is possible to identify through this analysis, which features are the most critical and which factors have the greatest impact on the decision.

In the case of this thesis work, the goal of this analysis is to understand how it is possible for the model to be trained with both DNS and RANS data to perform well although its inputs are conflicting.

6.1 Machine Learning Interpretability Methods

The interpretability methods for machine learning can be classified according to [36] into intrinsically interpretable methods and post hoc/model-agnostic interpretation methods. The first tends to reduce the complexity of the model, such as creating a simplified structure, and the second applies methods to analyze the model after the training (post hoc), such as the features permutation method.

According to [38], model-agnostic explanation systems are preferred for :

- The model flexibility: the interpretation can work with multiple machine learning models (random forests and deep neural networks);
- The explanation flexibility: not limited by a certain form of explanations. It can be used both a linear formula or a feature importance plot;
- The representation flexibility: the explanation of the model must be able to use a different representation of features than the one used by the model.

Model-agnostic interpretation can be further divided into global methods, which describe how features affect the prediction on average, and local methods which aim to explain individual predictions. Since only local methods were applied in this thesis work all the effort will be into explaining this typology. Commonly used local interpretation methods are the following:

- Feature importance: this method analyzes the importance of each feature to the final decision of the problem. Techniques such as permutation or ablation can be used. This category also includes analysis using Shapley values, which will be the ones primarily used in this thesis work;

- Partial dependence plot: shows the marginal that one or multiple features have on the model outcome. It can be useful to identify non-linear relationships between the feature and the model output;
- Local surrogate models (LIME): explain a prediction by replacing the complex model with a local surrogate model;
- Scoped rules (anchors): explains individual predictions on any black box model by finding a decision rule that "anchors" at the decision rule sufficiently. The decision rule, in this case, should be found i.e. (age>70 and sex = male).

The Shapley value analysis was chosen to perform the interpretability analysis of the model that will be described in section 6.2 mainly because it makes it possible to understand each feature's impact in the model predictions. Also in section 6.2 will be made a list explaining the reasons for this choice in more detail.

6.2 Shapley Value Theory

The Shapley value algorithm is a perturbation-based approach based on a cooperative game theory concept which allows the contribution to the outcome to be fairly distributed among the participants [39]. In the machine learning concept, the Shapley Value algorithm allows the contribution of the prediction to be distributed between all the input features. The goal of this analysis is therefore to understand how to distribute the output of the network in such a way that inputs are rewarded according to how much they contributed to that prediction.

According to [38] and [39] the Shapley values for the j-feature are ϕ_j, which represents the contribution to the outcome, are computed as follows:

$$\phi_j(\text{val}) = \sum_{S \subseteq \{1,\ldots,p\} \setminus \{j\}} \underbrace{\frac{|S|!(p-|S|-1)!}{p!}}_{\text{Weights}} \underbrace{(\text{val}(S \cup \{j\}) - \text{val}(S))}_{\text{Marginal Contributions } M_j} \tag{6.1}$$

where val() is the function that can be a neural network model with certain inputs or a played game; S is a subset of features; p is the total number of features and j is the single feature of which the Shapley value should be computed. The marginal contribution is defined as:

$$M_j = (\text{val}(S \cup \{j\}) - \text{val}(S))$$

To understand better an easy example is formulated below. Imagine a game in which 3 players (A, B, C) participate and which produces $100 as the outcome. One wants to determine through Shapley values the contribution of each player to this value. To compute these values, the game is computed with all the possible permutations of players (A, B, C, A+B, B+C, A+C and C+B) which produced different outcomes listed in Figure 6.1.

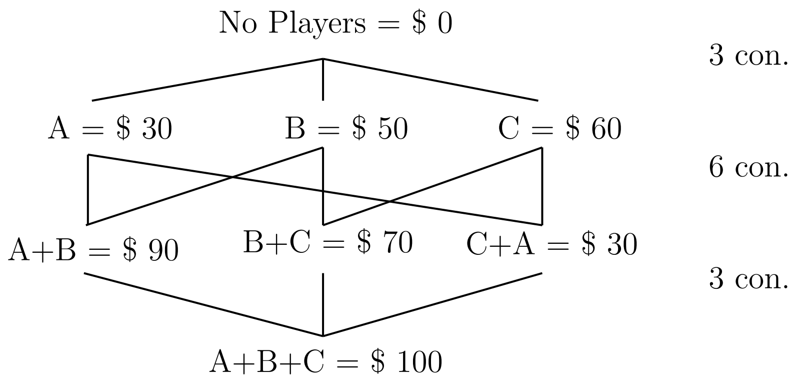


Figure 6.1: Shapley value example. It can be seen all the possible inputs permutation and the game outcome with that inputs. The lines represent the possible connection between the inputs in order to compute the marginal contribution.

The marginal contribution for the feature A can be computed by subtracting the game outcome with a set that contains A and a set without, as can be seen in Equation 6.2.

$$\begin{cases} M_{A1} = \text{val}(A) - \text{val}(0) = 30 - 0 = 30 \\ M_{A2} = \text{val}(A+B) - \text{val}(B) = 90 - 50 = 40 \\ M_{A3} = \text{val}(A+C) - \text{val}(C) = 30 - 60 = -30 \\ M_{A4} = \text{val}(A+B+C) - \text{val}(B+C) = 100 - 70 = 30 \end{cases} \tag{6.2}$$

The four marginal contributions for A should be summed according to the weights. The weights are proportional to the number of connections. The number of connections can be computed in Figure 6.1. In conclusion, the Shapley value for the feature A is:

$$\phi_A = \frac{1}{3}30 + \frac{1}{6}(40 - 30) + \frac{1}{3}30 = \$21.6 \tag{6.3}$$

the same can be done for B and C:

$$\phi_B = \frac{1}{3}50 + \frac{1}{6}(0 + 60) + \frac{1}{3}70 = \$51.6 \tag{6.4}$$

$$\phi_C = \frac{1}{3}60 + \frac{1}{6}(0 + 20) + \frac{1}{3}10 = \$26.6 \tag{6.5}$$

One of the most important properties of the Shapley value is the efficiency for what the sum of the Shapley value for all the features should be the model output with all the features inside. In the previous example can be observed that:

$$\phi_A + \phi_B + \phi_C = \text{val(A+B+C)} = \$100$$

Three other properties can be observed: the symmetry which says that if two features contribute equally in all possible coalitions their Shapley value should be the same; the dummy which says that if the Shapley value of a feature is zero it means that adding or subtracting this feature to a coalition doesn't change the outcome, and the last one the additivity which means that Shapley value can be summed.

The efficiency property can be summarized as starting from a point where no players are playing, in the example is 0, it can arrive at the situation where every player is in the game just summing the contributions of every single player. What was previously stated can be seen in Figure 6.2.

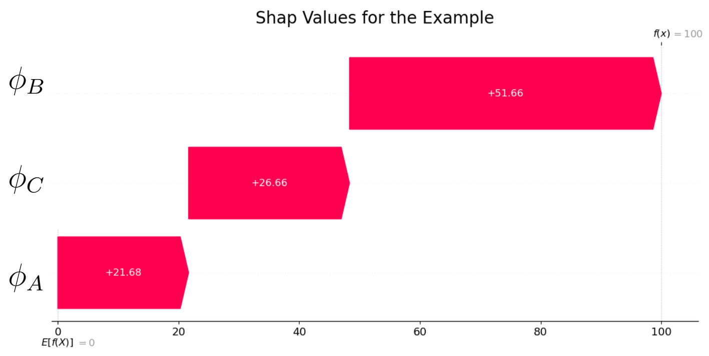


Figure 6.2: Shapley value efficiency properties. Starting from the condition that no player is in the game with outcome 0, the condition that all players are in the game with outcome 100 is reached by simply summing the Shapley values obtained.

What has been formulated for a cooperative game can also be said for a machine learning model in which the players turn out to be the features and the game turns out to be the model. Moving from the collaborative game theory to the neural network framework, the main problem is removing a feature in the machine learning model. In fact, feature removal is not possible in a mathematical function. It is not possible to re-train the network since the new re-trained model will be not the model to be explained. What is done to overcome this problem is instead of deleting a value is to replace the value to be deleted with a background value. This value can be zero, the mean, or the median of the dataset.

Another major problem of the Shapley value algorithm is the computational cost which requires 2^p evaluation of the network where p is the number of features. There are some algorithms that perform sampling to avoid this problem. Slundberg

[40] implemented this in the most famous Python library "SHAP"[1] and Castro' methods [41] has been implemented in the Captum[2] Python library. It is clear that if sampling is applied the efficiency property will not be completely respected.

6.3 Shapley Values Results

The Shapley algorithm explained in section 6.2 was applied to the neural network trained for this thesis work in particular the $\alpha = 1$ network of chapter 4. This network it is observed in chapter 5 to produce some very good results with both DNS and RANS inputs Figure 5.4. The interpretability algorithm was used to understand how the network manages to perform well with such different inputs. Specifically, an effort was made to look at how much reliance is placed on features (DNS and RANS). Furthermore, it was observed which features represent more importance by varying the Prandtl number in the channel flow simulation.

To recap the structure of the dataset considering only the 1D channel flow is highlighted in Table 3.2 in which is shown that the number of simulations is 34 (17 DNS + 17 RANS) and the computational points for each of them are 1000. This led to have 34 000 computational points in the datasets. The 34 simulations differ in the Re_τ and Pr number. By deciding the Re_τ and the Pr, 1000 computational points are selected for the DNS case and 1000 for the RANS one.

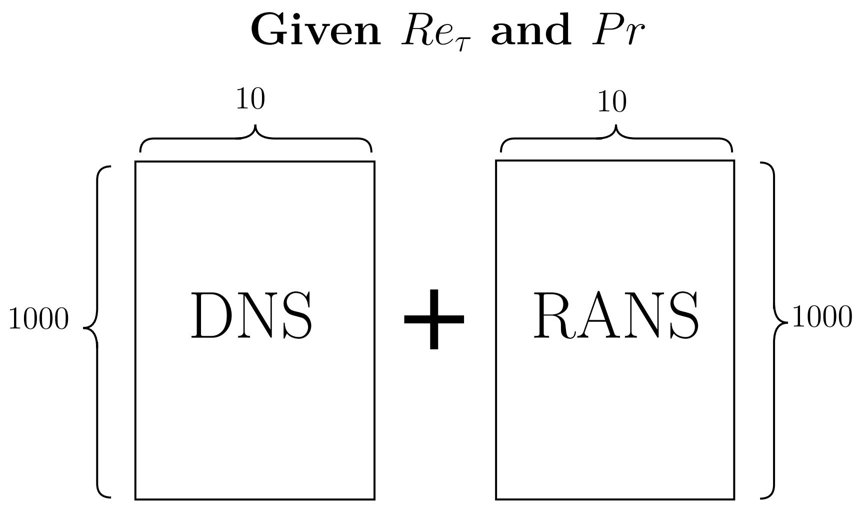


Figure 6.3: Dataset size for the Shapley Value analysis. In particular given a Re_τ and a Pr there are 1000 computational points for each simulation.

[1]See: https://github.com/slundberg/shap

[2]See: https://github.com/pytorch/captum

The analysis was done by grouping the features since firstly the total number of features (p) was very high and would have made the Shapley value algorithm too slow (network evaluations $\propto 2^p$) and secondly because with all those features it would have been difficult to interpret the results. Teams of features were created in according to the results obtained with PCA in Figure 3.11 and can be observed in Table 6.1. They were divided into 4 groups: features related to the momentum part; features related to the thermal part; features related to the closure thermal part and the tensors that play a separate role as they enter in the last part of the network (Figure 2.3). These four groups are defined in Table 6.1 in all their features.

Shapley Valus Teams of Features

Momentum	**Thermal**	**RANS Closure**	**Tensors**
π_1, π_2, π_3	Pr, R	π_5	T_1, T_2, T_3, T_4
π_4, Re_τ, ν_{root}	π_7, ∇T	π_6	T_5, T_6, T_7, T_8

Table 6.1: Features grouping for the Shapley value analysis. Every feature in the corresponding group is ablated when the group is ablated. The feature definition can be found in Table 2.2.

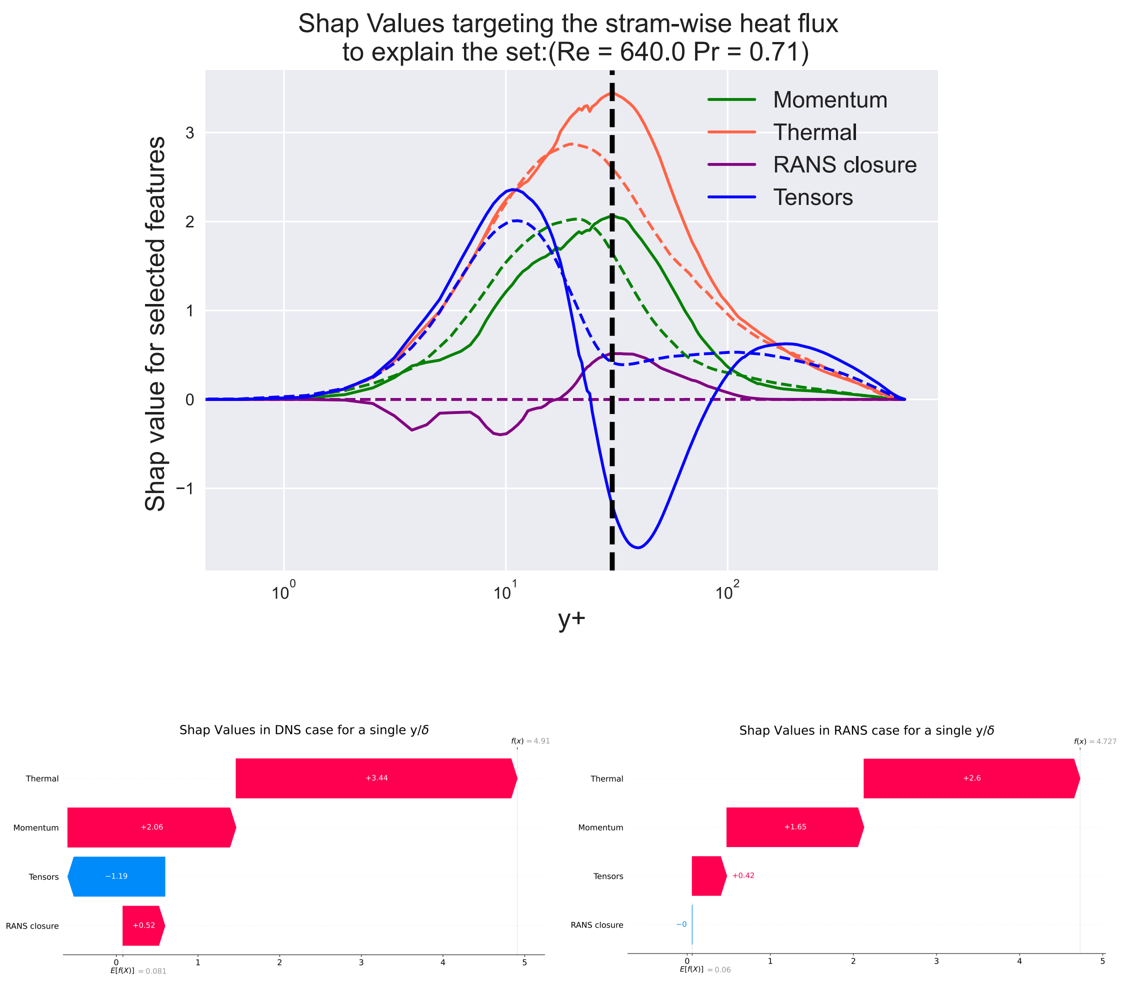


Figure 6.4: On top are represented the Shapley values for the $Re_\tau = 640$ and $Pr = 0.71$ simulation. The Shapley values computed with DNS inputs are in solid lines and the ones computed with RANS inputs are in dashed lines. The figure on the bottom shows the contribution to the stream-wise heat flux (Shapley values) in the location of the dashed black line. It can be observed that the sum of the contribution, starting from the background ($E[f(x)]$) where all features are neglected allows reaching the value for the stream-wise turbulent heat flux ($f(x)$).

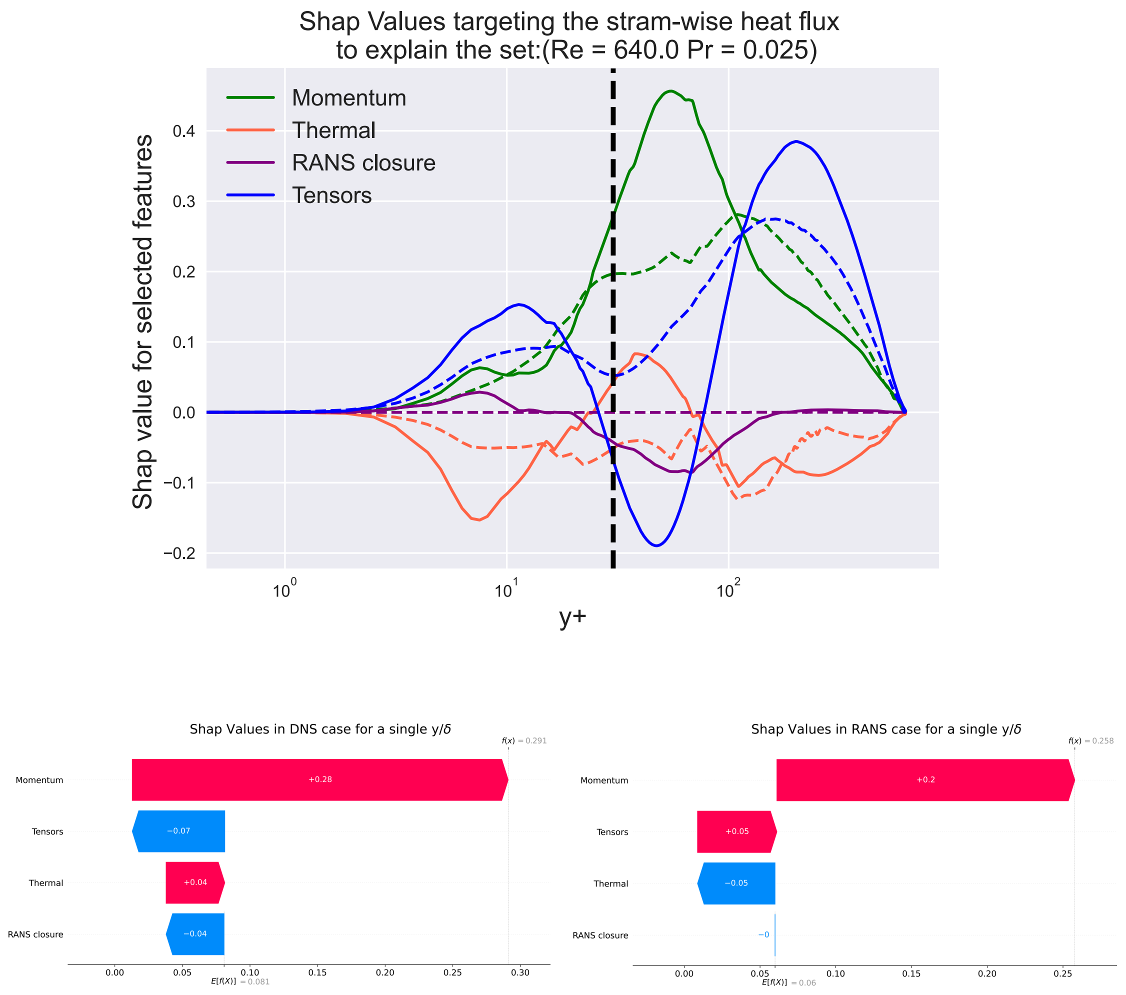


Figure 6.5: On top are represented the Shapley values for the $Re_\tau = 640$ and $Pr = 0.025$ simulation. The Shapley values computed with DNS inputs are in solid lines and the ones computed with RANS inputs are in dashed lines. The figure on the bottom shows the contribution to the stream-wise heat flux (Shapley values) at the location of the dashed black line. It can be observed that the sum of the contribution, starting from the background ($E[f(x)]$) where all features are neglected allows reaching the value for the stream-wise turbulent heat flux ($f(x)$).

The Shapley values are represented in Figure 6.4 and 6.5. The first representation is for $Pr = 0.71$ and the second one is for $Pr = 0.025$. Both of the plots are made with $Re_\tau = 640$. In these representations, the Shapley values are computed for both DNS (solid line) and RANS (dashed line) inputs. In section 6.2 was explained the efficiency principle of the Shapley values which states that the sum of the Shapley values is equal to the model prediction. To prove this, on the bottom of Figure 6.4

and 6.5 is plotted the cumulative sum of the Shapley values at a constant $y+$. The final sum at this coordinate represents the stream-wise turbulent heat flux.
From the previous charts it can be stated as follows:

- The thermal part is less important in the lower Prandtl number for two main reasons: the first concerns the non-normalization of the Shapley values since are physical values (efficiency property) and the second concerns the increase in the thermal conductivity with the decrease of the Pr which reduced the temperature fluctuations and the temperature gradients;

- The Shapley values computed with RANS inputs are always lower than the DNS ones at the same coordinate $y+$. This behavior is driven by RANS inputs being less precise than DNS ones. This has led the network to adapt to these inaccuracies and trust them less;

- The Shapley values computed with RANS inputs for the features responsible for the momentum closure (dashed purple line) are always zero. This means that they are useless for contributing to the heat flux but can serve as a classifier of the network;

- Comparing the two cases it can be seen that for $Pr = 0.71$ the part of the most significant influence of both thermal and momentum is at the same point. This behavior does not occur for $Pr = 0.025$. This is due to the fact that the closer the Prandtl is to one the more similarity between the momentum and the thermal part is obtained. This behavior seems to reflect the lack of accuracy between thermal and momentum fields at low Prandtl numbers

- In the $Pr = 0.025$ thermal Shapley value undergoes a double bending. This behavior can be explained by the fact that the lower the Prandtl, the smoother the temperature profile becomes with a delayed maximum slope compared to high Prandtl values. The gradient, therefore, decreases and shifts slightly. The result of this is that near the wall the thermal part does not bring a positive contribution to the heat flux (negative Shapley values) and the peak shifts slightly;

- For the $Pr = 0.025$ all the Shapley values seem more confusing, a sign that the network is having more trouble predicting a correct heat flux value anyway.

It is also interesting to observe the behavior of the wall-normal component in the heat flux which is represented in Figure 6.6 and 6.7.

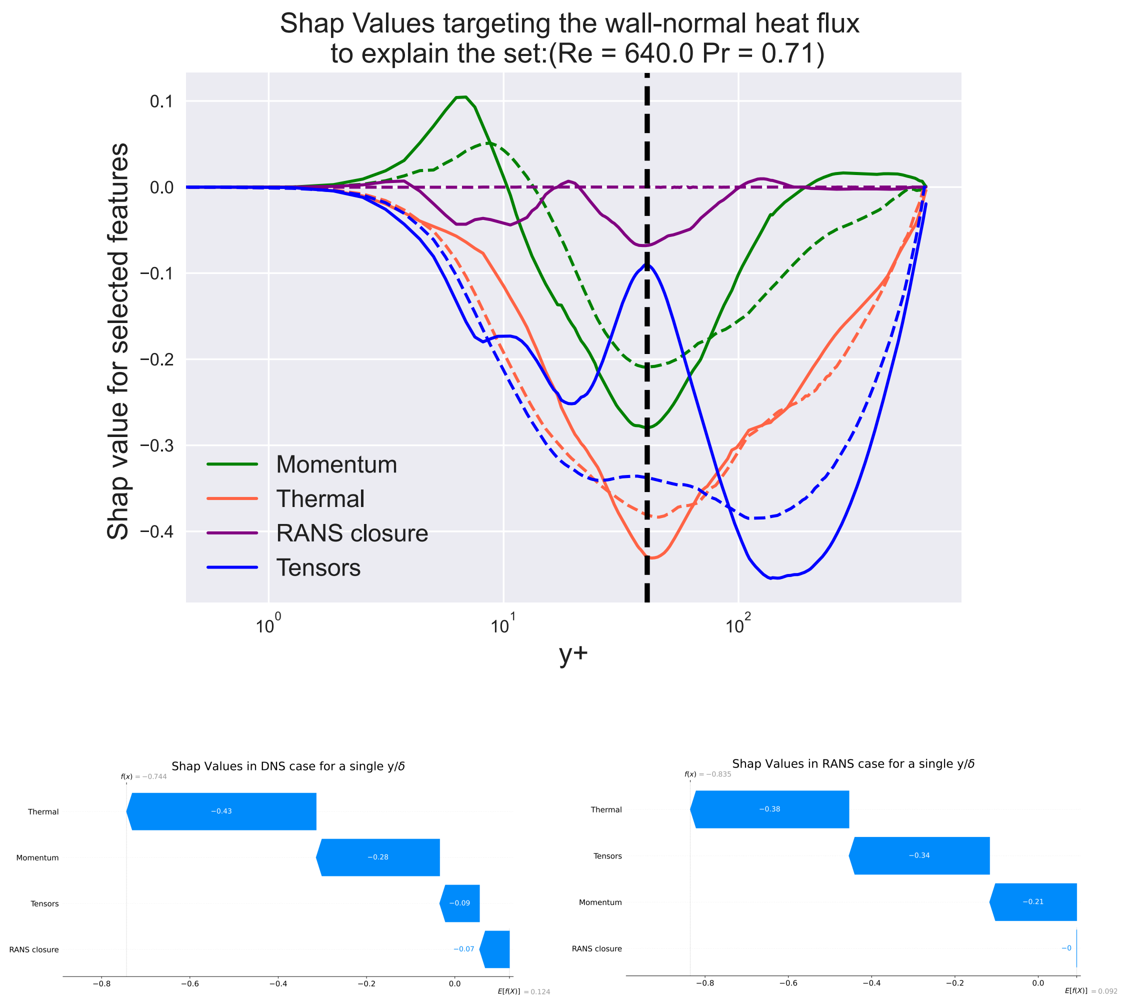


Figure 6.6: On top are represented the Shapley values for the $Re_\tau = 640$ and $Pr = 0.71$ simulation. The Shapley values computed with DNS inputs are in solid lines and the ones computed with RANS inputs are in dashed lines. The figure on the bottom shows the contribution to the wall-normal heat flux (Shapley values) at the location of the dashed black line. It can be observed that the sum of the contribution, starting from the background ($E[f(x)]$) where all features are neglected allows reaching the value for the wall-normal turbulent heat flux ($f(x)$).

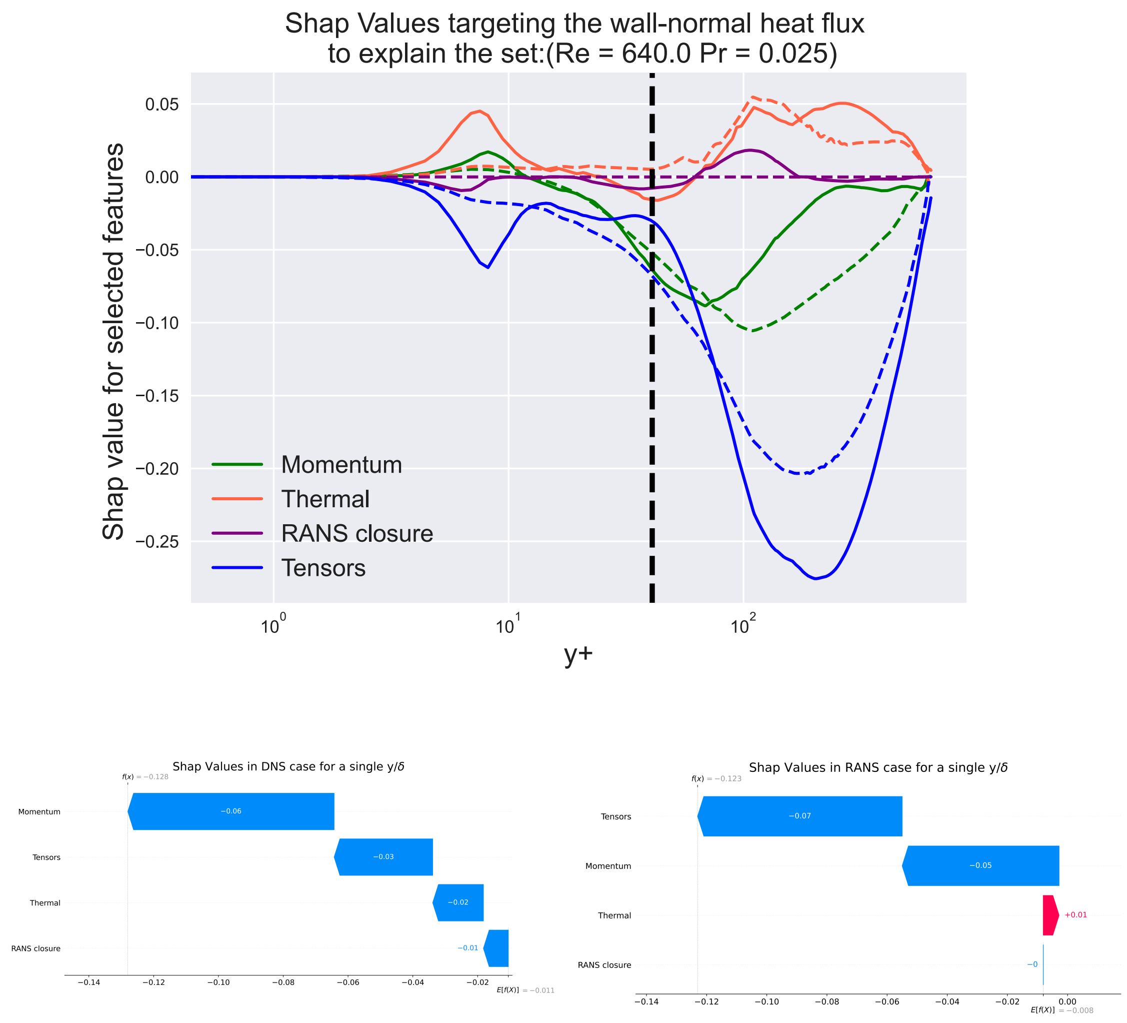


Figure 6.7: On top are represented the Shapley values for the $Re_\tau = 640$ and $Pr = 0.71$ simulation. The Shapley values computed with DNS inputs are in solid lines and the ones computed with RANS inputs are in dashed lines. The figure on the bottom shows the contribution to the wall-normal heat flux (Shapley values) at the location of the dashed black line. It can be observed that the sum of the contribution, starting from the background $(E[f(x)])$ where all features are neglected allows reaching the value for the wall-normal turbulent heat flux $(f(x))$.

Figure 6.6 and 6.7 represents the wall-normal heat flux. From these figures can be stated as follows:

- For high Prandtl values always the thermal part is of paramount importance. This is due to the reason explained above;

- The RANS part always remains more damped than the DNS part as had also happened with the stream-wise heat flux. This is due to the greater inaccuracy of these inputs;
- The zone of greatest momentum and thermal importance for high Pr remains confined around the same point while for low Prandtl values, it occurs in two different zones as had also been seen for the stream-wise heat flux;
- The greater importance of the tensor group is to be investigated more. To carry this out they should be divided by making the computation time increase exponentially;
- Also for these cases, the network has more difficulty predicting the wall-normal heat flux for low Pr than for higher Pr. It can be seen that the Shapley values are much more oscillatory at low Prandtl.

This analysis was very useful in understanding the importance of the inputs at different channel flow coordinates but especially at different Prandtl numbers. Future work could be to partition the inputs at the origin to actually calculate the importance of k, ϵ, U, etc... This will definitely require higher computational capacity as the number of permutations will increase exponentially. Another possibility may be to proceed through the calculation of Shapley values in a simplified manner as proposed in [41].

6.4 Layer Attribution Analysis

Layer activation analysis is a method used to interpret predictions made by a machine learning model (very commonly used in deep learning). This procedure allows the analysis of the activation pattern of one or more layers.

What has been done in this section is to analyze the last layer of the network to evaluate how it changes given DNS and RANS inputs. To accomplish this, the Captum library was used.

In Figure 6.8 is represented the network chart to remind the structure and to display the location of the analyzed layer. Specifically, as can be seen in Figure 6.8 and previously mentioned in subsection 2.2.2; the network is composed of two branches: the first with all the π_i and Re; the second with the Pr input. The two branches are subsequently multiplied in order to produce the neural network output (blue box in Figure 6.8).

The first analysis carried out is on the red box of Figure 6.8 which is the output of the Prandtl branch. The chart is expected to be only one, since between DNS and RANS the Prandtl number value does not change. This can be seen in Figure 6.9.

The second analysis carried out is on the blue box of Figure 6.8 which is the multiplication of the two branches' output. In this case, there are two representations since it varies between DNS and RANS.

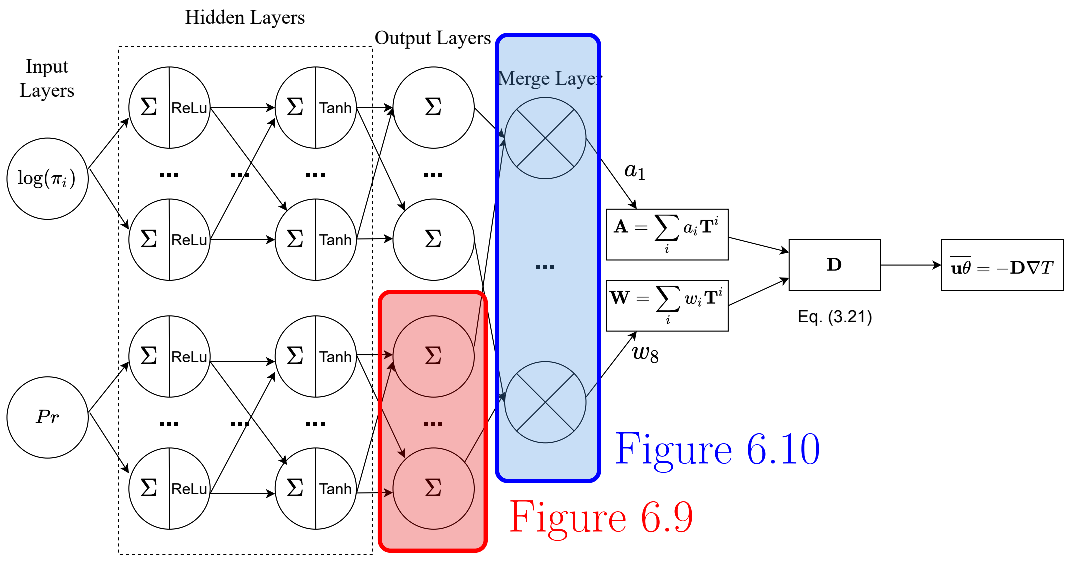

Hidden Layers
Input Layers
Output Layers
Merge Layer
$\log(\pi_i)$
Pr
ReLu
Tanh
Σ
a_1
w_8
$\mathbf{A} = \sum_i a_i \mathbf{T}^i$
$\mathbf{W} = \sum_i w_i \mathbf{T}^i$
D
Eq. (3.21)
$\overline{\mathbf{u}\theta} = -\mathbf{D}\nabla T$
Figure 6.10
Figure 6.9

Figure 6.8: Network visualization to understand which layers were analyzed. The red box is the Prandtl batch output and the blue box is representing the neural network output.

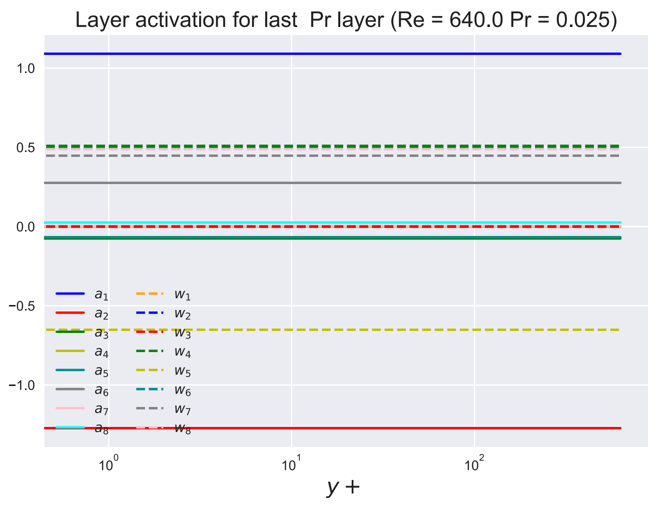
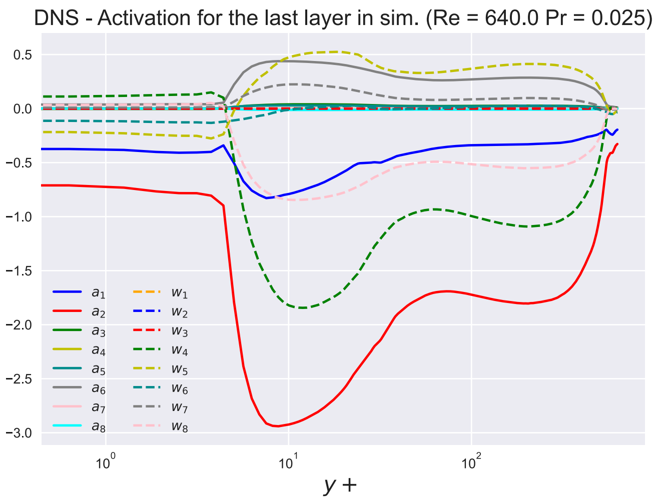


Figure 6.9: Layer output at the Prandtl branch (red box of Figure 6.8). It is the same with both DNS and RANS inputs.



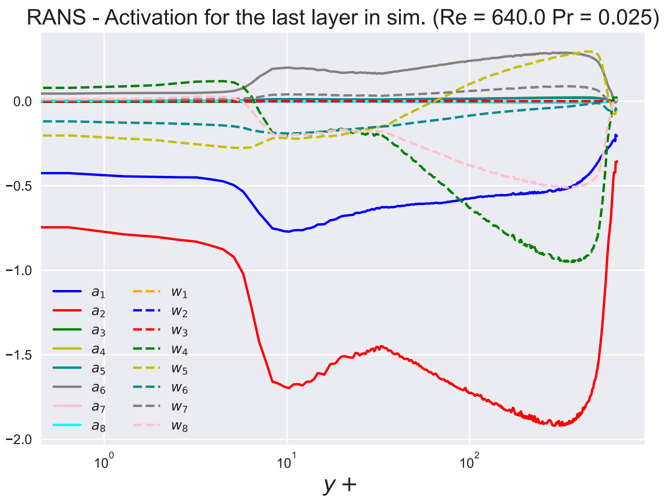


Figure 6.10: Neural network output (blue box of Figure 6.8). On top the output with DNS inputs and, on the bottom, with RANS inputs.

It can be said from Figure 6.9 that the Prandtl branch sets firstly to zero all the a_i coefficients except for a_1, a_2 and a_6 and secondly it leaves for the $\boldsymbol{W}$ tensor (defined in Equation 2.22) the components w_4, w_5, w_6, w_7 and w_8. Thus, the role of the Prandtl number is to eliminate, dampen or amplify the output of the other branch of the network.

In Figure 6.10 it is possible to observe the components responsible for the symmetric part of the tensor $\boldsymbol{D}$ which are a_i and the ones responsible for the asymmetric part which are w_i. The following can be stated from this last figure:

- It is quite clear that a_1 and a_2 are the most important components which bring high importance to the tensors $\boldsymbol{T_1}$ and $\boldsymbol{T_2}$;

- In the near-wall region a_1 is less precise with RANS inputs than with DNS inputs for the reason that is linked to $\boldsymbol{T_1}$ and, $\boldsymbol{T_1}$, is highly dependent on the RANS momentum turbulence model;
- The asymmetric part of $\boldsymbol{D}$ represented by the coefficients w_i is mostly related to the rotation. This can be observed by looking at the most important w_i component that is w_4 which is connected to $\boldsymbol{T_4}$ composed by the vorticity tensor $\boldsymbol{\Omega}$. It is also noted that all the non-zero w_i components are the ones related to a $\boldsymbol{T_i}$ which contains $\boldsymbol{\Omega}$ except for w_5;
- The output of the network computed with RANS inputs has the w_i components not present in the near-wall region. This behavior is due to the momentum turbulence model. This can be the biggest difference in computing the two types of inputs in the output layer.

Conclusions and Perspectives

This thesis study introduced and presented a new data-driven model to study and model the turbulent heat flux, with special emphasis on low Prandtl number fluids. The model previously trained in Fiore et al. [1], which was used as a starting point, seemed to be too sensitive to the defects of the momentum modelling, especially all the models that apply the Boussinesq hypothesis like the $k-\epsilon$. This behavior was mainly brought by the fact that the model was trained on high-fidelity (DNS) data only.

Therefore, while keeping the same input formulation as the previous work, a database of inputs from Reynolds Averaged Navier Stokes (RANS) simulations using the Boussinesq hypothesis, was added to the database. It was found that the two databases were not identical, but differed mainly in certain inputs, i.e. π_3, π_5 and π_6 that are functions, as expected, of the anisotropic part of the Reynolds stresses. Thus, due to the conflicting nature of certain inputs, it was necessary to reformulate the loss function to take into account both DNS and RANS inputs, thereby adjusting a hyperparameter, called α, that weighs the importance of the two sub-losses.

To tune this parameter, a Pareto front was created and it was found that the best trade-off is obtained for $\alpha = 1$, which means using an equal weight between the DNS and RANS losses.

One- and two-dimensional flow configurations in the low Prandtl number regime were validated *a priori* and *a posteriori*. The achieved results showed that the new data-driven model ($\alpha = 1$) performs well both inside and outside of the training conditions range. Moreover, it is very important to point out that the model was able to overcome the obstacles encountered by the previous network ($\alpha = 0$) like the applicability of the network with Boussinesq based momentum models. In fact, as can be seen in the chart below (Figure 7.11), very satisfactory results were obtained for the different heat flux components with both DNS and RANS ($k-\epsilon$) inputs.

Finally, a model interpretability analysis with Shapley values was performed to

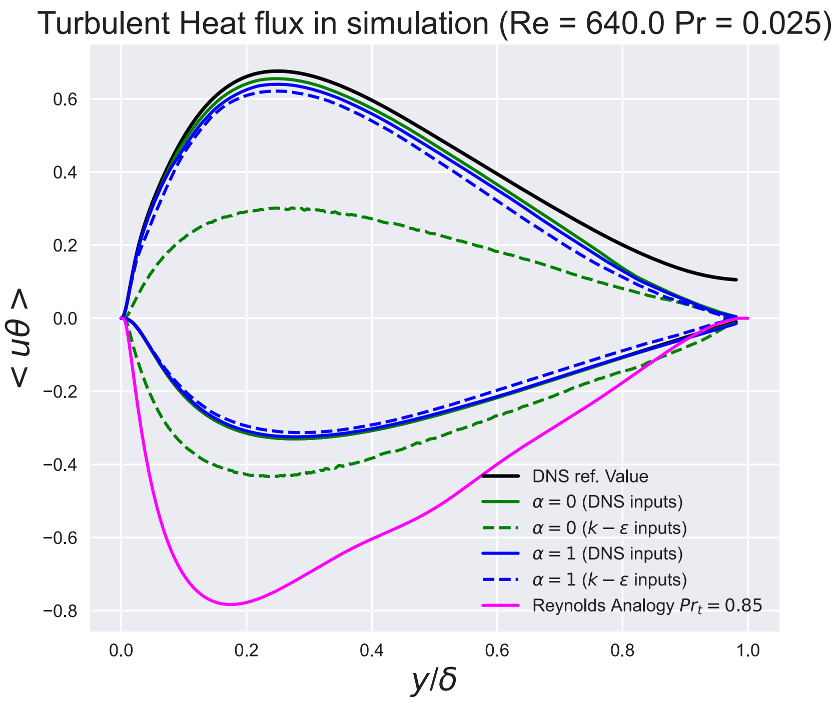


Figure 7.11: Stream-wise heat flux ($\overline{u\theta}$) on the top, and wall-normal heat flux ($\overline{v\theta}$) on the bottom for $Re_\tau = 640$ and $Pr = 0.025$. The black line is the DNS reference value, the blue line is the value with model trained in this thesis work ($\alpha = 1$), the green line is the value with the previous model ($\alpha = 0$), and, the pink line, is the value with the Reynolds analogy. It is clear that the $\alpha = 1$ model (blue lines) manages to work well with both DNS (solid line) and RANS (dashed line) inputs.

understand how the model behaved and which features were found to be the most important with DNS and RANS inputs. Indeed, as expected, it was found that the thermal part (temperature, thermal variance, and thermal dissipation rate) becomes less important as the Prandtl number decreases, since thermal conductivity and diffusivity increase, mitigating temperature fluctuations and damping gradients. This analysis also showed that the model relies less on RANS inputs than DNS ones. This can be explained by the fact that the model has learned to trust them less as they are less accurate.

Hence, further efforts and future research may be directed towards:

- A different formulation of the inputs for the presence of singularities in some of them. Indeed, this new formulation should prevent them from tending to very high values at the wall, as could be observed with the present formulation. Therefore, this behavior caused the dataset to have outliers and high skewness, which affected the performance of the model;

- Increasing the dataset size with more two-dimensional and three-dimensional flows. This can actually lead to a greater generalization of the model;
- The implementation of a new Robust Principal Component Analysis (RPCA), which may be able to reduce the spatial size of the dataset by losing less variance and allowing a new network to be trained with a reduced number of components. In fact, with the current algorithm, 22.5% of the total variance is lost;
- A new interpretability analysis that considers not only features (π_i) or team of features but directly the outputs of CFD simulations such as U, k, ϵ, and so on. This will be very instructive as it will allow to really understand which physical quantity induces certain heat flux values and how the model makes use of them.

Bibliography

[1] Matilde Fiore, Lilla Koloszar, Clyde Fare, Miguel Alfonso Mendez, Matthieu Duponcheel, and Yann Bartosiewicz. «Physics-constrained machine learning for thermal turbulence modelling at low Prandtl numbers». In: *International Journal of Heat and Mass Transfer* 194 (2022), p. 122998. ISSN: 0017-9310. DOI: https://doi.org/10.1016/j.ijheatmasstransfer.2022.122998. URL: https://www.sciencedirect.com/science/article/pii/S0017931022004719 (cit. on pp. ii, 22–28, 59, 62, 63, 66, 97).

[2] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. «Machine Learning for Fluid Mechanics». In: *Annual Review of Fluid Mechanics* 52.1 (2020), pp. 477–508. DOI: 10.1146/annurev-fluid-010719-060214. URL: https://doi.org/10.1146/annurev-fluid-010719-060214 (cit. on pp. 1, 2, 4, 10–13).

[3] Michele Milano and Petros Koumoutsakos. «Neural Network Modeling for Near Wall Turbulent Flow». In: *Journal of Computational Physics* 182.1 (2002), pp. 1–26. ISSN: 0021-9991. DOI: https://doi.org/10.1006/jcph.2002.7146. URL: https://www.sciencedirect.com/science/article/pii/S0021999102971469 (cit. on p. 3).

[4] Brendan Tracey, Karthik Duraisamy, and Juan Alonso. «Application of Supervised Learning to Quantify Uncertainties in Turbulence and Combustion Modeling». In: *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. DOI: 10.2514/6.2013-259. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2013-259. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2013-259 (cit. on p. 3).

[5] Karthikeyan Duraisamy, Ze J. Zhang, and Anand Pratap Singh. «New Approaches in Turbulence and Transition Modeling Using Data-driven Techniques». In: *53rd AIAA Aerospace Sciences Meeting*. DOI: 10.2514/6.2015-1284. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2015-1284. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2015-1284 (cit. on p. 3).

[6] Ze Jia Zhang and Karthikeyan Duraisamy. «Machine Learning Methods for Data-Driven Turbulence Modeling». In: *22nd AIAA Computational Fluid Dynamics Conference*. DOI: `10.2514/6.2015-2460`. eprint: `https://arc.aiaa.org/doi/pdf/10.2514/6.2015-2460`. URL: `https://arc.aiaa.org/doi/abs/10.2514/6.2015-2460` (cit. on p. 3).

[7] V. Cherkassky and F.M. Mulier. *Learning from Data: Concepts, Theory, and Methods*. IEEE Press. Wiley, 2007. ISBN: 9780470140512. URL: `https://books.google.be/books?id=IMGzP-IIaKAC` (cit. on p. 3).

[8] *What are neural networks?* URL: `https://www.ibm.com/topics/neural-networks` (cit. on p. 5).

[9] Pragati Baheti. *Activation Functions in Neural Networks*. Oct. 2022. URL: `https://www.v7labs.com/blog/neural-networks-activation-functions` (cit. on pp. 5–8).

[10] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. «Activation functions in neural networks». In: *towards data science* 6.12 (2017), pp. 310–316 (cit. on p. 6).

[11] Jeremy Jordan. *Setting the learning rate of your neural network*. Mar. 2023. URL: `https://www.jeremyjordan.me/nn-learning-rate/` (cit. on p. 10).

[12] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. *Generative Adversarial Networks*. 2014. arXiv: `1406.2661 [stat.ML]` (cit. on p. 13).

[13] C. Drygala, B. Winhart, F. di Mare, and H. Gottschalk. «Generative modeling of turbulence». In: *Physics of Fluids* 34.3 (Mar. 2022), p. 035114. DOI: `10.1063/5.0082562`. URL: `https://doi.org/10.1063%2F5.0082562` (cit. on p. 13).

[14] Taehyuk Lee, Junhyuk Kim, and Changhoon Lee. «Turbulence control for drag reduction through deep reinforcement learning». In: *Phys. Rev. Fluids* 8 (2 Feb. 2023), p. 024604. DOI: `10.1103/PhysRevFluids.8.024604`. URL: `https://link.aps.org/doi/10.1103/PhysRevFluids.8.024604` (cit. on p. 13).

[15] Stephen B. Pope. *Turbulent Flows*. Cambridge University Press, 2000. DOI: `10.1017/CBO9780511840531` (cit. on pp. 14, 15).

[16] Joel H. [2] Ferziger. *Computational methods for fluid dynamics / J.H. Ferziger, M. Peric*. eng. 2nd rev. ed. Berlin: Springer, 1999. ISBN: 978-3-540-65373-8 (cit. on pp. 14, 15).

[17] A. Shams, A. De Santis, L.K. Koloszar, A. Villa Ortiz, and C. Narayanan. «Status and perspectives of turbulent heat transfer modelling in low-Prandtl number fluids». In: *Nuclear Engineering and Design* 353 (2019), p. 110220. ISSN: 0029-5493. DOI: https://doi.org/10.1016/j.nucengdes.2019.110220. URL: https://www.sciencedirect.com/science/article/pii/S0029549319302407 (cit. on p. 16).

[18] Xianwen Li, Xingkang Su, Long Gu, Lu Zhang, and Xin Sheng. «Numerical study of low pr flow in a bare 19-rod bundle based on an advanced turbulent heat transfer model». In: *Frontiers in energy research* 10 (2022). ISSN: 2296-598x. DOI: 10.3389/fenrg.2022.922169. URL: https://www.frontiersin.org/articles/10.3389/fenrg.2022.922169 (cit. on pp. 17, 20).

[19] S. Manservisi and F. Menghini. «A CFD four parameter heat transfer turbulence model for engineering applications in heavy liquid metals». In: *International Journal of Heat and Mass Transfer* 69 (2014), pp. 312–326. ISSN: 0017-9310. DOI: https://doi.org/10.1016/j.ijheatmasstransfer.2013.10.017. URL: https://www.sciencedirect.com/science/article/pii/S0017931013008776 (cit. on pp. 17, 19).

[20] Jorge E Bardina, Peter G Huang, and Thomas J Coakley. «Turbulence modeling validation, testing, and development». In: (1997) (cit. on p. 18).

[21] A. Shams et al. «A collaborative effort towards the accurate prediction of turbulent flow and heat transfer in low-Prandtl number fluids». In: *Nuclear Engineering and Design* 366 (2020), p. 110750. ISSN: 0029-5493. DOI: https://doi.org/10.1016/j.nucengdes.2020.110750. URL: https://www.sciencedirect.com/science/article/pii/S0029549320302442 (cit. on p. 21).

[22] A. J. M. Spencer and R. S. Rivlin. «The theory of matrix polynomials and its application to the mechanics of isotropic continua». In: *Archive for Rational Mechanics and Analysis* 2.1 (Jan. 1958), pp. 309–336. DOI: 10.1007/BF00277933 (cit. on p. 24).

[23] Hiroshi Kawamura, H. Abe, and K. Shingai. «DNS of turbulence and heat transport in a channel flow with different Reynolds and Prandtl numbers and boundary conditions». In: *Proceedings of the 3rd International Symposium on Turbulence, Heat and Mass Transfer* (Jan. 2000) (cit. on pp. 31, 32, 68).

[24] Iztok Tiselj, Robert Bergant, Borut Mavko, Ivan Bajsic´, and Gad Hetsroni. «DNS of Turbulent Heat Transfer in Channel Flow With Heat Conduction in the Solid Wall ». In: *Journal of Heat Transfer* 123.5 (Mar. 2001), pp. 849–857. ISSN: 0022-1481. DOI: 10.1115/1.1389060. eprint: https://asmedigitalcollection.asme.org/heattransfer/article-pdf/123/5/849/5733971/849_1.pdf. URL: https://doi.org/10.1115/1.1389060 (cit. on p. 31).

[25] Jure Oder, Afaque Shams, Leon Cizelj, and Iztok Tiselj. «Direct numerical simulation of low-Prandtl fluid flow over a confined backward facing step». In: *International Journal of Heat and Mass Transfer* 142 (2019), p. 118436 (cit. on pp. 31, 34).

[26] Ian T. Jolliffe and Jorge Cadima. «Principal component analysis: a review and recent developments». In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374.2065 (2016), p. 20150202. DOI: `10.1098/rsta.2015.0202`. URL: `https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2015.0202` (cit. on pp. 39, 40).

[27] Karl Pearson F.R.S. «LIII. On lines and planes of closest fit to systems of points in space». In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: `10.1080/14786440109462720`. URL: `https://doi.org/10.1080/14786440109462720` (cit. on p. 39).

[28] Harold Hotelling. «Analysis of a complex of statistical variables into principal components.» In: *Journal of Educational Psychology* 24 (1933), pp. 498–520 (cit. on p. 39).

[29] Konstantinos I Diamantaras and Sun Yuan Kung. *Principal component neural networks: theory and applications*. John Wiley & Sons, Inc., 1996 (cit. on p. 39).

[30] Bernhard Flury. *Common principal components & related multivariate models*. John Wiley & Sons, Inc., 1988 (cit. on p. 39).

[31] Kalyanmoy Deb and Deb Kalyanmoy. *Multi-Objective Optimization Using Evolutionary Algorithms*. USA: John Wiley & Sons, Inc., 2001. ISBN: 047187339X (cit. on p. 54).

[32] BV Venkatesh and H Nilsson. «Tutorial of convective heat transfer in a vertical slot». In: *proceedings of CFD with opensource software* (2016) (cit. on p. 65).

[33] Giovanni Maria Carlomagno and Andrea Ianiro. «Thermo-fluid-dynamics of submerged jets impinging at short nozzle-to-plate distance: A review». In: *Experimental Thermal and Fluid Science* 58 (2014), pp. 15–35. ISSN: 0894-1777. DOI: `https://doi.org/10.1016/j.expthermflusci.2014.06.010`. URL: `https://www.sciencedirect.com/science/article/pii/S0894177714001514` (cit. on p. 73).

[34] M. Duponcheel and Y. Bartosiewicz. «Direct Numerical Simulation of Turbulent Heat Transfer at Low Prandtl Numbers in Planar Impinging Jets». In: *International Journal of Heat and Mass Transfer* 173 (2021), p. 121179. ISSN: 0017-9310. DOI: https://doi.org/10.1016/j.ijheatmasstransfer.2021.121179. URL: https://www.sciencedirect.com/science/article/pii/S0017931021002829 (cit. on pp. 73–76).

[35] Andrea De Santis, Agustin Villa Ortiz, Afaque Shams, and Lilla Koloszar. «Modelling of a planar impinging jet at unity, moderate and low Prandtl number: Assessment of advanced RANS closures». In: *Annals of Nuclear Energy* 129 (2019), pp. 125–145. ISSN: 0306-4549. DOI: https://doi.org/10.1016/j.anucene.2019.01.039. URL: https://www.sciencedirect.com/science/article/pii/S0306454919300465 (cit. on p. 75).

[36] Haitz Sáez de Ocáriz Borde, David Sondak, and Pavlos Protopapas. «Convolutional neural network models and interpretability for the anisotropic reynolds stress tensor in turbulent one-dimensional flows». In: *Journal of Turbulence* 23.1-2 (Nov. 2021), pp. 1–28. DOI: 10.1080/14685248.2021.1999459. URL: https://doi.org/10.1080%2F14685248.2021.1999459 (cit. on pp. 81, 82).

[37] Tim Miller. *Explanation in Artificial Intelligence: Insights from the Social Sciences*. 2018. arXiv: 1706.07269 [cs.AI] (cit. on p. 81).

[38] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. 2nd ed. 2022. URL: https://christophm.github.io/interpretable-ml-book (cit. on pp. 82, 83).

[39] Lloyd S. Shapley. *A Value for N-Person Games*. Santa Monica, CA: RAND Corporation, 1952. DOI: 10.7249/P0295 (cit. on p. 83).

[40] Scott M Lundberg and Su-In Lee. «A Unified Approach to Interpreting Model Predictions». In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 4765–4774. URL: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf (cit. on p. 86).

[41] Javier Castro, Daniel Gomez, and Juan Tejada. «Polynomial calculation of the Shapley value based on sampling». In: *Computers & Operations Research* 36 (May 2009), pp. 1726–1730. DOI: 10.1016/j.cor.2008.04.004 (cit. on pp. 86, 93).

Appendix A

Boussinesq's hypothesis effect in input calculation

A.1 Effect on π_6

The neural network scalars inputs are shown in Table 2.2. Only π_6 was taken as an example to demonstrate that the application of the Boussinesq hypothesis leads to values that are always zero in the case of RANS inputs in a 1D test case.

Assuming a 2D flow the tensors $\mathbf{b}$, $\mathbf{S}$ and $\boldsymbol{\Omega}$ that represent respectively the turbulence tensor, the strain-rate tensor, and the vorticity tensor have the following definition:

$$\mathbf{b} = \begin{pmatrix} b_{11} & b_{12} & 0 \\ b_{12} & b_{22} & 0 \\ 0 & 0 & b_{33} \end{pmatrix}$$

$$\mathbf{S} = \begin{pmatrix} S_{11} & S_{12} & 0 \\ S_{12} & S_{22} & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\boldsymbol{\Omega} = \begin{pmatrix} 0 & \Omega_{12} & 0 \\ -\Omega_{12} & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

It can be proved that π_6 can be written as:

$$\pi_6 = \{\mathbf{b}\mathbf{S}\boldsymbol{\Omega}\}\,\frac{k^2}{\epsilon^2} \tag{A.1}$$

$$\pi_6 = \Omega_{12}(b_{12}S_{11} + b_{22}S_{12} - b_{11}S_{12} - b_{12}S_{22})\frac{k^2}{\epsilon^2} \tag{A.2}$$

Isolating only the tensors trace and applying the definition of the tensor:

$$\{\mathbf{bS\Omega}\} = \frac{1}{2}\left(\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}\right)\left[\frac{\overline{uv}}{k}\left(\frac{\partial u}{\partial x}\right) + \frac{1}{2}\left(\frac{\overline{vv}}{k} - \frac{2}{3}\right)\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) + \right.$$
$$\left. - \frac{1}{2}\left(\frac{\overline{uu}}{k} - \frac{2}{3}\right)\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) - \frac{\overline{uv}}{k}\frac{\partial v}{\partial y}\right]$$

Grouping:

$$\{\mathbf{bS\Omega}\} = \frac{1}{2}\left(\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}\right)\left[\frac{\overline{uv}}{k}\left(\frac{\partial u}{\partial x}\right) - \frac{\overline{uv}}{k}\left(\frac{\partial v}{\partial y}\right) + \right.$$
$$\left. + \underbrace{\frac{1}{2}\left(\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}\right)\left(\frac{\overline{vv}}{k} - \frac{2}{3} - \frac{\overline{uu}}{k} + \frac{2}{3}\right)}_{=0}\right]$$

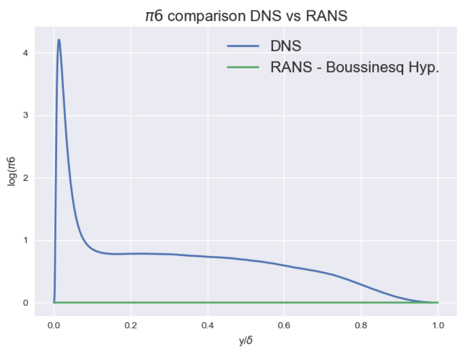
The third term is equal to zero since in the 2D flow case for the Boussinesq's hypotheses implies that $\overline{vv} - \overline{uu} = 0$

The trace equation for π_6 can be reduced in this way:

$$\{\mathbf{bS\Omega}\} = \frac{1}{2}\left(\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}\right)\left[\frac{\overline{uv}}{k}\left(\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}\right)\right] \tag{A.3}$$

- If the flow is 1D in the channel flow case, the only derivative of the velocity that is non-zero is the $\frac{\partial u}{\partial y}$ and, as a consequence of this, it can be observed that Equation A.3 is always zero. It can be noted that in this situation the DNS value is different from zero since $\overline{vv} - \overline{uu} \neq 0$ and the π_6 value computed with a momentum closure that applies the Boussinesq's hypotheses such as $k - \epsilon$ will always be zero (Figure A.1);

- If the flow is 2D the term in Equation A.3 is not always zero but differs greatly from the DNS. In particular with a momentum closure that applies the Boussinesq hypotheses such as $k - \epsilon$ will be much lower than the DNS value (Figure A.2).



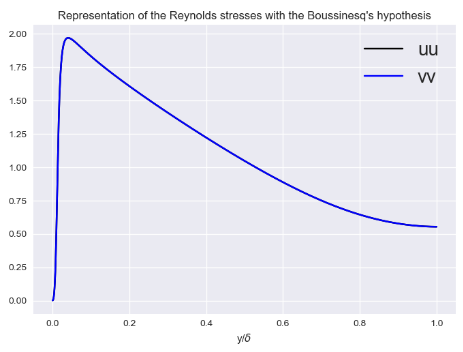


Figure A.1: The figure on the left shows the difference between π_6 in the DNS case and the RANS with $k - \epsilon$ turbulence model. The right figure shows the Boussinesq's hypothesis effect on the Reynolds stress $(\overline{uu} - \overline{vv}) = 0$

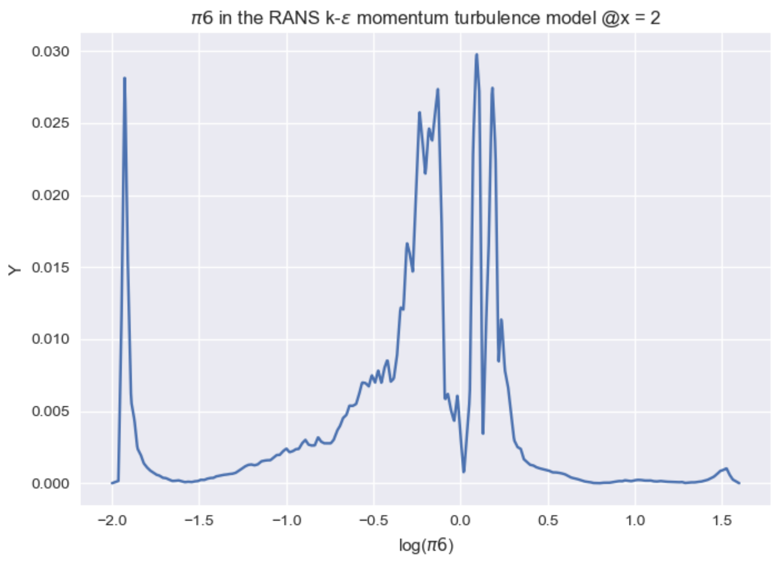


Figure A.2: A representation of π_6 in case of a 2D flow with $k - \epsilon$ turbulence model. As we see the value is not zero as mentioned in Equation A.3. The chart was made in the backward facing step at a constant value x = 2

A.2 Effect on π_1 and π_2 in the 1D flow

In this section, it is shown that π_1 and π_2 have the same values (in both DNS and RANS datasets) if the 1D flow hypothesis is applied.

Starting from the definition seen in Table 2.2 of π_1 and π_2 :

$$\begin{cases} \pi_1 = \dfrac{k^2}{\epsilon^2}\{\boldsymbol{S^2}\} \\ \\ \pi_2 = \dfrac{k^2}{\epsilon^2}\{\boldsymbol{\Omega^2}\} \end{cases}$$

it is necessary to prove that in this case:

$$\{\boldsymbol{S^2}\} \overset{!}{=} \{\boldsymbol{\Omega^2}\} \tag{A.4}$$

where $\{\boldsymbol{S}\}$ and $\{\boldsymbol{\Omega}\}$ are defined in Equation 2.15.

The following can be obtained:

$$\begin{aligned} \{\boldsymbol{S^2}\} &= S_{11}^2 + 2S_{12}^2 + S_{22}^2 \\ \{\boldsymbol{\Omega^2}\} &= -2\Omega_{12}^2 \end{aligned}$$

applying the definition of S and Ω and the fact that only $\frac{\partial u}{\partial y} \neq 0$:

$$\begin{aligned} \{\boldsymbol{S^2}\} &= \left(\frac{\partial u}{\partial x}\right)^2 + 2\frac{1}{4}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 &&= \frac{1}{2}\left(\frac{\partial u}{\partial y}\right)^2 \\ \{\boldsymbol{\Omega^2}\} &= -2\frac{1}{4}\left(\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}\right)^2 &&= -\frac{1}{2}\left(\frac{\partial u}{\partial y}\right)^2 \end{aligned}$$

since at π_1 and π_2 is applied the absolute value before the logarithmic function the same value of π_1 and π_2 will be obtained in case of 1D inputs.

Appendix B

Pareto Front Algorithm

Algorithm 1 Pareto Front algorithm made to obtain the best α value

```
1:  procedure ParetoFront($\boldsymbol{\alpha}, \mathcal{M}, \boldsymbol{X}$)
2:      ▷ $\boldsymbol{\alpha}$ is a vector containing all the possible $\alpha$
3:      ▷ $\boldsymbol{\alpha} = [0.001,\ 0.01,\ 0.1,\ 1,\ 10,\ 100,\ 1000]$
4:      ▷ $\mathcal{M}$ is the neural network structure
5:      ▷ $\boldsymbol{X}$ is the input matrix
6:      $t_f \leftarrow$ max epochs number
7:      for $\alpha_k$ in $\boldsymbol{\alpha}$ do
8:          $\boldsymbol{X}_{1_{\text{DNS}}} \leftarrow$ get_1D_dns($\boldsymbol{X}$)
9:          $\boldsymbol{X}_{2_{\text{DNS}}} \leftarrow$ get_2D_dns($\boldsymbol{X}$)
10:         $\boldsymbol{X}_{1_{\text{RANS}}} \leftarrow$ get_1D_rans($\boldsymbol{X}$)
11:         $\boldsymbol{X}_{2_{\text{RANS}}} \leftarrow$ get_2D_rans($\boldsymbol{X}$)
12:         while $t$ not $t_k$ do
13:             $t \leftarrow t + 1$
14:             $\mathcal{L}_{1_{\text{DNS}}} \leftarrow$ loss_fnc($\mathcal{M}(\boldsymbol{X}_{1_{\text{DNS}}})$)
15:             $\mathcal{L}_{2_{\text{DNS}}} \leftarrow$ loss_fnc($\mathcal{M}(\boldsymbol{X}_{2_{\text{DNS}}})$)
16:             $\mathcal{L}_{1_{\text{RANS}}} \leftarrow$ loss_fnc($\mathcal{M}(\boldsymbol{X}_{1_{\text{RANS}}})$)
17:             $\mathcal{L}_{2_{\text{RANS}}} \leftarrow$ loss_fnc($\mathcal{M}(\boldsymbol{X}_{2_{\text{RANS}}})$)
18:             $\mathcal{L} \leftarrow \mathcal{L}_{1_{\text{DNS}}} + \mathcal{L}_{2_{\text{DNS}}} + \alpha(\mathcal{L}_{1_{\text{RANS}}} + \mathcal{L}_{2_{\text{RANS}}})$
19:         end while
20:     end for
21:     return best $\alpha_k$
22: end procedure
```

Appendix C

OpenFoam Network Implementation

This appendix explains in detail how to implement a network written through the PyTorch library in the OpenFoam framework.

As explained in section 5.2, the network was coded with Python language and OpenFoam is written in C/C++. Since the Tensorflow APIs were already implemented in OpenFoam the network was previously converted from Pytorch to Tensorflow and implemented into the OpenFoam solver. The implementation steps are the following:

1. Conversion of the PyTorch network to a Tensorflow network passing through Onnx as can be seen in Figure 5.6;

2. Validation of the Tensorflow model;

3. Installing the Tensorflow APIs and compiling them into OpenFoam.

C.1 Neural Network conversion

In this section is explained how to convert the PyTorch network structure file to the Tensorflow one. There is no direct conversion from PyTorch to Tensorflow but it is necessary to pass through Onnx.

C.1.1 Installing the Python Environment

The first step is the python environment installing. It can be installed with the following command and the requirements file shown below.

Terminal window

```
pip install -r /path/to/requirements.txt
```

File: requirements.txt

```
absl-py==1.0.0
astor==0.8.1
captum==0.5.0
certifi==2021.5.30
charset-normalizer==2.0.12
cloudpickle==2.0.0
coverage==4.0.3
cycler==0.11.0
Cython==0.29.24
cytoolz==0.11.0
dask==2021.3.0
dataclasses==0.8
decorator==5.1.1
docopt==0.6.2
gast==0.5.3
google-pasta==0.2.0
grpcio==1.35.0
h5py==2.10.0
idna==3.4
imagecodecs==2020.5.30
imageio==2.9.0
Keras-Applications==1.0.8
Keras-Preprocessing==1.1.2
kiwisolver==1.3.1
Markdown==2.6.9
matplotlib==3.3.4
mkl-fft==1.3.0
mkl-random==1.1.1
mkl-service==2.3.0
networkx==2.5
numpy==1.19.2
olefile==0.46
onnx==1.10.1
onnx-tf==1.1.2
Pillow==8.3.1
pip==21.2.2
pipreqs==0.4.11
protobuf==3.18.0
pyparsing==3.0.4
pyreadline==2.1
python-dateutil==2.8.2
PyWavelets==1.1.1
PyYAML==5.4.1
requests==2.27.1
```

```
scikit-image==0.17.2
scipy==1.5.3
setuptools==58.0.4
six==1.16.0
tensorboard==1.14.0
tensorflow==1.14.0
tensorflow-estimator==1.14.0
termcolor==1.1.0
tifffile==2021.3.17
toolz==0.11.2
torch==1.8.2
torchaudio==0.8.2
torchvision==0.9.2
tornado==6.1
typing_extensions==4.1.1
urllib3==1.26.14
Werkzeug==1.0.1
wheel==0.37.1
wincertstore==0.2
wrapt==1.11.2
yarg==0.1.9
```

After creating the environment one can proceed to the conversion of the network.

C.1.2 PyTorch to Tensorflow network

The first thing is to initialize an empty PyTorch model, for example in this way recalling the THFNet is a class based on the *torch.nn.Module* and the input variables represent the network layers' sizes and dimensions:

```
model_pytorch = THFNet(D_in, D_in_pr, H, H_pr, D_out, D_out_pr)
```

The network must be fed by the weights and biases found in the training phase with *load_state_dict* function on the *torch.nn.Module* class:

```
model_pytorch.load_state_dict(torch.load(os.path.join(os.getcwd(), '
    Net_50_1.0.pt'), map_location=torch.device('cpu')))
```

The next step is the creation of some dummy inputs used to check the network dimensions. It is strongly recommended to create an input and output name vector. After that, the PyTorch model can be converted into an Onnx model with the function *torch.onnx.export*.

```
input_names = ['input_1'] + ['input_2'] + ['input_3']
```

```
output_name = ['out1'] + ... + ['out16']
torch.onnx.export(model_pytorch, dummy_input, "torch_noise1_Exported.onnx
    ", verbose=True,input_names=input_names, output_names=output_names,
    export_params=True)
```

The Onnx model should be loaded and the chunk dimensions should be modified in order to receive any dimension on the computational domain. This is made to allow the model to work with every mesh size. The new model is now saved as an ".onnx" model.

```
model_onnx.graph.input[0].type.tensor_type.shape.dim[0].dim_param = '?'
model_onnx.graph.input[1].type.tensor_type.shape.dim[0].dim_param = '?'
model_onnx.graph.input[2].type.tensor_type.shape.dim[0].dim_param = '?'

onnx.save(model_onnx, "torch_noise1_Onnx.onnx")
onnx_model_2 = onnx.load("torch_noise1_Onnx.onnx")
```

The last thing is the conversion from Onnx to Tensoflow which can be done as follows:

```
tf_rep = prepare(onnx_model_2, 'cpu')
tf_rep.export_graph('Alpha1Noise.pb')
```

The Tensorflow (".pb") is now exported and ready to be implemented into the OpenFoam framework.

C.2 Validation of the Tensoflow model

It is strongly recommended to check the exported Tensorflow model in order to see if all the dimensions and the structure has been correctly converted. To make this validation the *Netron website* can be used. On this website, it is possible to upload the ".pb" file and see the network structure. It is important to check:

- The network structure;
- The network inputs/outputs names;
- The inputs dimensions in particular if there is a "?" otherwise the network wont go with different mesh sizes.

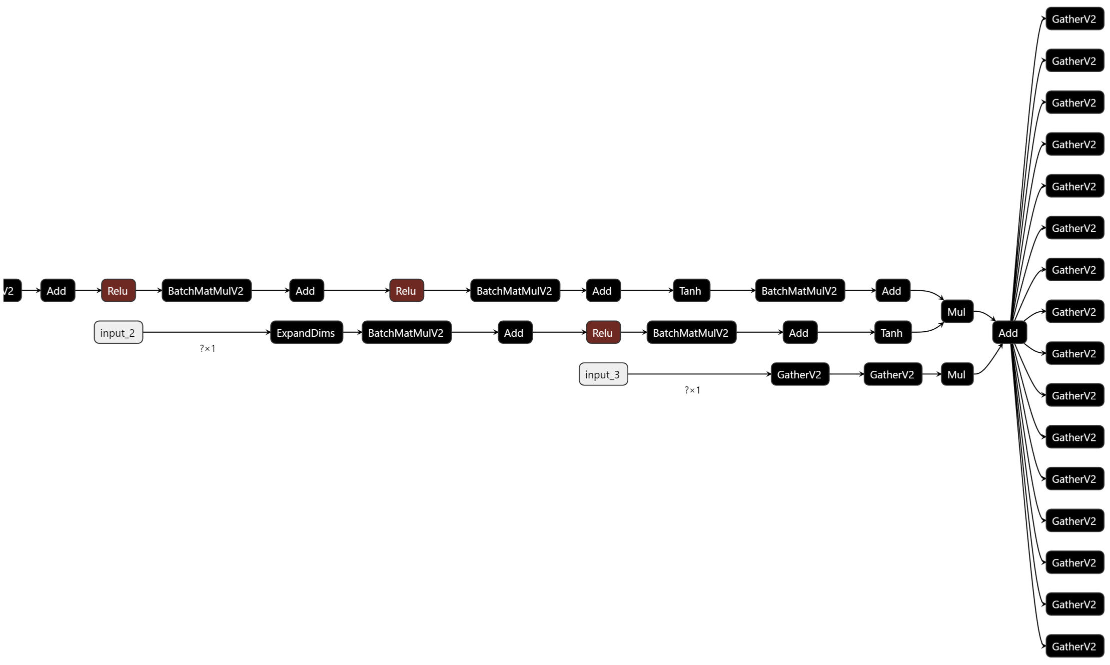


Figure C.1: A section of the Tensorflow exported network view in Netron. It's important to notice the "?" in the input size

C.3 Compiling the Tensorflow APIs in OpenFoam

To enable the network to work in OpenFoam the Tensorflow C-APIs should be compiled in OpenFoam. Firstly the APIs should be downloaded from the Tensorflow website *tensorflow.org/install/lang_c*. An easy way to test the APIs is to create a "TensorfloC" folder in which the previously downloaded folders are extracted inside, create a ".cpp" test script that can be found in Listing C.3, open a terminal window in the "TensorflowC" folder, and run the following bash command:

Terminal window

```
g++ -I/path/to/extracted/TensorflowC/include -L/path/to/extracted/
    TensorflowC/lib test.cpp -ltensorflow

./a.out
```

"Hello from TensorFlow C library version XXXX" should appear in the terminal view.
The test file can be the following:

test.cpp

```
#include <stdio.h>
#include <tensorflow/c/c_api.h>

int main() {
  printf("Hello from TensorFlow C library version %s\n", TF_Version()
    );
  return 0;
}
```

The last step remains to edit the *.bashrc* file and to implement the APIs in OpenFoam. In the *.bashrc* file add the following lines:

/.bashrc

```
export LIBRARY_PATH=$LIBRARY_PATH:path/to/TensorflowC/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:path/to/TensorflowC/lib

ulimit -s unlimited
```

In the *Make/options* file add the following lines:

.../Make/options

```
EXE_INC = \
     -I path/to/TensorflowC/include \
     -I path/to/TensorflowC/include/tensorflow/c \
...
LIB_LIBS = \
     -ltensorflow \
     -lstdc++ \
...
```

Now a turbulence thermal model can be created in the OpenFoam source folder:

Terminal window

```
cd ~/OpenFOAM/xx-8/src/ThermophysicalTransportModels/turbulence/
    Name_OF_Your_Model
```