



HAL
open science

Deep Wall Models for Aerodynamic Simulations

Michele Romanelli

► **To cite this version:**

Michele Romanelli. Deep Wall Models for Aerodynamic Simulations. Mathematics [math]. Université de Bordeaux, 2024. English. NNT : 2024BORD0358 . tel-04933669

HAL Id: tel-04933669

<https://hal.science/tel-04933669v1>

Submitted on 6 Feb 2025

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE PRÉSENTÉE
POUR OBTENIR LE GRADE DE
DOCTEUR
DE L'UNIVERSITÉ DE BORDEAUX
ÉCOLE DOCTORALE
MATHÉMATIQUES ET INFORMATIQUE
MATHÉMATIQUES APPLIQUÉES ET CALCUL SCIENTIFIQUE

Par **Michele ROMANELLI**

Deep learning wall laws for aerodynamic simulations

Sous la direction de : **Héloïse BEAUGENDRE**
Co-directeur : **Michel BERGMANN**

Soutenue le 12 décembre 2024

Membres du jury :

Mme. Héloïse BEAUGENDRE	Professeur des Universités	Bordeaux INP	Directrice
M. Michel BERGMANN	Directeur de Recherche	Université de Bordeaux IMB	Co-directeur
M. Raphaël LOUBÈRE	Directeur de Recherche	Université de Bordeaux IMB	President
M. Pierre SGAUT	Professeur des Universités	Université d'Aix-Marseille	Rapporteur
M. Lars DAVIDSON	Professeur des Universités	Université de technologie Chalmers	Rapporteur
Mme. Camilla FIORINI	Maîtresse de conférences	CNAM	Examinatrice
M. Denis SIPP	Directeur de Recherche	ONERA	Encadrant
M. Samir BENEDDINE	Ingénieur de Recherche	ONERA	Encadrant
M. Ivan MARY	Ingénieur de Recherche	ONERA	Encadrant

Acknowledgments

This PhD research was conducted within the framework of Chaire PROVE, co-funded by the Nouvelle-Aquitaine region.

The journey of a PhD is a challenging one, requiring the collaboration of many people, and it certainly wouldn't have been possible without their support. This process demands perseverance, continuous encouragement, insightful feedback, and, of course, an endless supply of caffeine. I am deeply grateful to everyone who helped make this work possible. Although it's impossible to name everyone who contributed over the past three years, I apologize for any unintentional omissions.

First and foremost, I would like to thank the reviewers, Pierre Sagaut and Lars Davidson, for their thorough evaluation of my thesis and their valuable feedback. I sincerely appreciate their willingness to read the entire manuscript.

I am also grateful to the examiners, Raphaël Loubère and Camilla Fiorini, for their expertise and thoughtful questions. Their feedback provided new perspectives and invaluable guidance for future research.

A special thank you goes to Samir for proposing this PhD topic and for sharing countless ideas and insights throughout this journey. His invaluable feedback and unwavering encouragement during our weekly meetings kept me motivated, especially during my most challenging moments.

I must apologize to Ivan for all the times I unexpectedly knocked on his office door with coding problems. I truly appreciate his patience and willingness to help, even when my visits were frequent and unannounced. Thank you, Ivan!

Thanks are also due to Denis for his endless flow of innovative ideas, often accompanied by formulas I didn't always understand. More importantly, his enthusiasm and passion for research made our discussions both inspiring and thought-provoking.

I would like to thank my PhD director, Héloïse, for her guidance and valuable insights throughout this work. Her expertise and feedback have been greatly appreciated.

I am also grateful to my co-director, Michel, for his advice and perspective, which have contributed to shaping this research.

I am sincerely grateful to the MASH team at ONERA and the Memphis team at Inria for welcoming me warmly and providing a supportive and stimulating research environment throughout my PhD.

I would also like to thank Angelo Iollo and everyone involved in the Chaire PROVE for the insightful discussions during our meetings. Presenting my work there was excellent training for my defense and I truly appreciate the experience, as well as the enjoyable moments shared afterward.

A huge thank you goes to Tanya for expertly navigating the maze of administrative paperwork over these past three years.

I want to extend my heartfelt gratitude to those I spent the most time with during my years at ONERA, who have become true friends. Thank you, Romain and Mathieu

L., for welcoming me into your office during my first days and for sharing invaluable tips that helped me navigate the early, confusing stages of being a newly started PhD student. Your support and patience made all the difference.

I would also like to thank all my colleagues at ONERA and my fellow PhD students, Bartolomeo, Pierre, Loïc, Hugo, Vianney, Alessandro, Mathieu S., Kevin, Julian, Arthur P., Arthur V., Raphael, Riccardo, Carmen, Laura P., Stefan, Jaime, Julien and Camille. I apologize if I have inadvertently forgotten anyone's name. Thank you for the countless moments shared over coffee breaks, ping-pong games or baby-foot matches. You all made this journey more enjoyable and helped me face its ups and downs with a smile.

Finally, I express my deepest gratitude to my family for their unwavering support, even when I made the bold and self-sacrificing decision to pursue a PhD. Thank you for always being there, even when I doubted myself.

And last but not least, I want to thank my girlfriend, Laura. Your support, patience and presence have made all the difference throughout this journey. I apologize for my bad moods, especially towards the end of my PhD, I know it wasn't easy.

Thank you all.

Contents

Acknowledgments	1
Contents	6
Acronyms	8
Introduction	9
Objectives	10
Structure of the thesis	11
1 State of the art	13
1.1 Introduction	14
1.2 Numerical methods for computational fluid dynamics	14
1.2.1 Reynolds Averaged Navier-Stokes Equations	14
1.2.2 Spalart-Allmaras turbulence model	16
1.2.3 Finite volume method	17
1.2.3.1 Convective flux discretization	18
1.2.3.1.1 Roe scheme	18
1.2.3.1.2 Third order MUSCL reconstruction	19
1.2.3.2 Viscous flux discretization	20
1.2.3.3 Numerical stencil	20
1.3 Boundary layer	21
1.3.1 Boundary layers	21
1.3.1.1 Types of boundary layer	22
1.3.1.2 Boundary layer separation	23
1.3.2 Turbulent boundary layer	23
1.3.2.1 Turbulent boundary layer equations	23
1.3.2.2 Self-similar solution of the velocity distribution	24
1.3.2.2.1 Inner layer	25
1.3.2.2.2 Outer layer	26
1.3.2.2.3 Overlap layer	26
1.3.2.3 Turbulent boundary layer structure	27
1.4 Wall modeling	28
1.4.1 Standard wall models	29
1.4.1.1 Differential wall models	30
1.4.1.2 Shear stress models	31
1.4.1.2.1 Analytical wall models	31
1.4.1.2.2 Integral wall models	32
1.4.2 Data-driven wall models	33
1.5 Neural Networks	34

1.5.1	Fully Connected Neural Networks	34
1.5.1.1	Activation functions	35
1.5.2	Training a neural network	36
1.5.2.1	Gradient back-propagation	36
1.5.2.2	Update of neural network parameters	39
1.5.2.3	Common gradient-based optimizers for neural networks	39
1.5.2.4	Neural network accuracy and architecture	40
1.5.2.5	Training techniques and issues	41
2	Methods and tools	45
2.1	Methodology and tools for data-driven wall models	45
2.1.1	Workflow of the data-driven wall model	46
2.1.2	Machine Learning Framework	46
2.1.2.1	Data Preprocessing	47
2.1.2.1.1	Non-dimensionalization	47
2.1.2.1.2	Normalization	47
2.1.2.1.3	Transformation	48
2.1.2.2	Neural network training	48
2.1.2.2.1	Neural network architecture	49
2.1.2.2.2	Loss function	49
2.1.2.2.3	Sample weighting	50
2.1.2.2.4	Optimizer and training strategy	51
2.1.3	CFD solver	53
2.1.3.1	Numerical approach	53
2.1.3.2	Neural network integration	53
2.2	Wall model strategy	54
2.2.1	Wall model strategy in a finite volume framework	54
2.2.2	Ghost cells approach	55
2.2.3	Model components	56
2.3	Main test cases	56
2.3.1	2D Bump case	56
2.3.1.1	Domain discretization	58
2.3.2	Airfoil case	58
2.3.2.1	Domain discretization	59
3	Data-driven wall models for RANS	63
3.1	Wall law formulation	64
3.1.1	Formulation for the wall tangent velocity evolution	65
3.1.2	Physical model for thermodynamic state and wall normal velocity field	65
3.1.2.1	Wall normal velocity	66
3.1.3	Near-wall Spalart-Allmaras modeling	66
3.2	Numerical implementation of the wall model	66
3.2.1	Iterative estimation of local wall shear stress	67
3.2.2	Wall model application	67
3.3	Flow configurations	68
3.4	Neural network implementation and training	71
3.4.1	Loss function definition	71
3.4.2	Neural network architecture and optimization	72
3.4.2.1	Optimization of the neural network architecture	73

3.4.2.2	Neural network architecture	74
3.4.3	Training and a priori results	75
3.5	Results	77
3.5.1	Test procedure	77
3.5.2	Global errors	78
3.5.3	Interpolation test results	78
3.5.4	Extrapolation cases	82
3.5.4.1	Flat plate case	83
3.5.4.2	Near separation case	83
3.5.4.3	Influence of dimensionless pressure gradient	83
3.5.5	Mass conservation	84
3.6	Conclusion	86
4	Efficient data-driven wall models for RANS	89
4.1	Wall law formulation	91
4.1.1	Dirichlet-To-Neumann formulation for the wall tangent velocity evolution	91
4.1.1.1	Additional parameters for Dirichlet-To-Neumann map	92
4.1.2	Physical model for thermodynamic state and wall normal velocity field	92
4.1.3	Near-wall Spalart-Allmaras modeling	93
4.2	Numerical implementation of the wall model	94
4.2.1	Wall model discretization	94
4.2.2	Wall model application	94
4.2.3	Numerical effect of Dirichlet-To-Neumann approach	95
4.3	Flow configurations	97
4.3.1	Bump flow case	97
4.3.1.1	Training and testing datasets	97
4.3.2	Airfoil case	98
4.4	Data-driven modeling	98
4.4.1	Neural networks	98
4.4.2	Neural network architecture	99
4.4.3	Dataset treatment	99
4.4.4	Loss function selection	101
4.4.5	Training strategy	102
4.4.6	Discussion on direct estimation and derivative computation	102
4.4.7	Discussion on single and multiple neural networks approach	103
4.5	Results	105
4.5.1	Results on bump geometry	105
4.5.1.1	Test procedure	105
4.5.1.2	Assessment of Spalart-Allmaras modeling strategy	107
4.5.1.3	Discussion on the additional parameters for DtN-map	108
4.5.1.4	Model validation and comparison with iterative approach	109
4.5.1.5	Interpolation in Reynolds number	112
4.5.1.6	Interpolation in bump height	112
4.5.1.7	Interpolation in bump height and Reynolds number	113
4.5.2	Test on the airfoil geometry	114
4.5.2.1	Test setup and procedure	115
4.5.2.2	Results	115
4.5.3	Numerical performances assessment	117

4.5.3.1	Model computational time assessment	117
4.5.3.2	Total convergence time assessment	120
4.6	Conclusion	121
5	Current development and future works	125
5.1	Optimization of computational performance	125
5.2	Detection of extrapolation conditions	126
5.2.1	Density-based outliers detection	127
5.2.2	Outliers detection Gilbert–Johnson–Keerthi algorithm	127
5.2.3	Discussion of application	129
5.3	Extension of validity domain to complex flow configurations	130
5.4	Immersed Boundary Method application	131
	Conclusion	133
	French summary / Résumé en français	137
	Bibliography	151

Acronyms

- Adam** Adaptive Moment Estimation. 40
- BL** boundary layer. 11, 14, 21–24, 27, 29, 47
- CFD** computational fluid dynamics. 9, 11, 14, 20, 28, 45–47, 49, 53, 64, 73, 83, 86, 87, 125, 133, 135, 137, 138, 140
- CNN** Convolutional Neural Network. 34
- CPU** Central Processing Unit. 125
- DES** Detached Eddy Simulation. 30
- DL** Deep Learning. 11, 14, 33
- DtN** Dirichlet-To-Neumann. 11, 90–92, 95, 98, 99, 122, 139
- ELU** Exponential Linear Unit. 35, 37, 42
- FastS** FAST Structured. 53, 138
- FCNN** Fully Connected Neural Network. 34–37, 40, 53
- FNN** Feedforward Neural Network. 34
- FVM** Finite Volume Method. 17, 18
- GJK** Gilbert–Johnson–Keerthi. 126–130, 135, 141
- GPU** Graphics Processing Unit. 125, 135, 141
- IBM** Immersed Boundary Method. 9, 10, 12, 29, 122, 125, 131, 132, 135, 137, 141
- KDE** kernel density estimation. 51, 102, 127
- Leaky ReLU** Leaky Rectified Linear Unit. 35, 37, 42
- LES** Large Eddy Simulation. 9, 30, 33, 135
- LSTM** Long Short-Term Memory Network. 34
- MAE** Mean Absolute Error. 36

- MAPE** Mean Absolute Percentage Error. 36, 52, 74, 75, 103–105
- ML** Machine Learning. 11, 14, 33, 34, 53
- MLP** Multilayer Perceptron. 34
- MSE** Mean Squared Error. 36, 72
- MSRE** Mean Squared Relative Error. 72
- NN** neural network. 10–12, 14, 33–36, 38–43, 47, 49–54, 64, 71–78, 83, 84, 86, 87, 102, 125–131, 133–135, 141
- NS** Navier-Stokes. 14, 23, 24, 30
- ODE** ordinary differential equation. 31
- PDE** partial differential equation. 31
- R-WM I.** RANS - Wall model interface. 54–56, 67, 77, 78, 80, 81, 86, 94, 107, 109, 110, 119, 122
- RANS** Reynolds Averaged Navier-Stokes. 9–12, 14, 16, 17, 24, 30–32, 46, 49, 53–55, 66–68, 70, 71, 73, 76–78, 80–83, 86, 90, 94, 97, 105–108, 110, 112–118, 120–122, 131, 133–135, 137–141
- ReLU** Rectified Linear Unit. 35, 37, 39, 42
- RL** Reinforcement Learning. 33
- RMSprop** Root Mean Square Propagation. 40
- RNN** Recurrent Neural Network. 34
- S.-A.** Spalart-Allmaras turbulence model. 16, 66, 80–83, 131, 134, 139, 140
- SGD** Stochastic Gradient Descent. 39, 40
- SGS** Sub-Grid Scale. 33
- TBL** turbulent boundary layer. 21, 23, 24, 27, 28, 31, 133, 135
- WM** wall model. 28–31, 125, 130–135
- ZPG** zero-pressure gradient. 32, 33

Introduction

In computational fluid dynamics (CFD), accurately predicting turbulent flows near solid boundaries is critical for a variety of industrial applications, including aerospace, automotive engineering, energy and manufacturing. The ability to carry out precise and rapid CFD simulations is vital for optimizing designs, evaluating performance and addressing conception challenges. These simulations are commonly used in industrial domain, however, their accuracy heavily depends on resolving near-wall flows, which play a crucial role in predicting phenomena such as drag, heat transfer, and flow separation [87]. This near-wall accuracy is essential for ensuring the reliability of results in critical engineering decisions.

Accurately capturing wall-bounded turbulent flows typically requires a dense mesh to resolve boundary layer profiles and steep gradients in the wall-normal direction. This significantly increases the computational load, strongly impacting the time and resources required for simulations. For instance, a wall-resolved Large Eddy Simulation (LES) of complex representative aeronautical application still remains far out-of-reach even for modern supercomputers [85]. Therefore, wall models become essential for applying LES in aeronautical and other high-Reynolds-number applications, where resolving all scales in the boundary layers is not affordable.

Therefore, Reynolds Averaged Navier-Stokes (RANS) simulations remain the standard in industrial applications due to their reduced computational cost. Current progresses in computational power make RANS able to handle complex industrial configurations. Nonetheless, further reducing the computational cost remains crucial for unsteady RANS simulations in many cases, for instance to study systems involving both very slow and very fast time scales (which may happen when a flow involves multiple instability phenomenon for instance, requiring to perform a high number of solver iterations) or for scenarios that require numerous simulations, such as parametric studies, optimization problems, flow control by reinforcement learning, etc. In these cases, wall models help alleviate grid resolution constraints by modeling the near-wall region instead of fully resolving it, thus lowering the associated computational cost without sacrificing significant accuracy.

Additionally, near-wall modeling is crucial for RANS simulations that utilize Immersed Boundary Methods (IBMs), where the interaction between the physical boundaries and the fluid within Cartesian computational grids is entirely represented by wall models. IBM is particularly advantageous in industrial design processes, offering a more straightforward meshing process of computational domains compared to body-fitted methods, allowing for efficient handling of complex geometries and evolving designs. This makes wall modeling indispensable in competitive environments where timely and accurate results are essential for innovation and optimization [75].

Therefore, the development and refinement of wall models remain a central focus in CFD research, as they are key to improving both accuracy and efficiency across a range of different turbulent flow simulations.

Wall modeling can be broadly categorized into two types: wall shear stress models and differential models. Wall shear stress models rely on simplified empirical formulations based

on the self-similar solution of the turbulent boundary layer's inner region or on the integral form of simplified transport equations. These models assume that the velocity profile in the near-wall region follows a predefined distribution and enforce the viscous constraint at the wall surface. They significantly reduce computational costs by bypassing the need to fully resolve the near-wall region's complex turbulence dynamics. However, these models may struggle to capture flow phenomena like separation, reattachment, or the effects of strong pressure gradients, where the standard assumptions break down. Differential wall models, on the other hand, solve simplified transport equations for quantities like velocity or turbulent kinetic energy near the wall. These models offer more flexibility and accuracy in capturing complex flow behaviors, such as those seen in adverse pressure gradient regions or separated flows, but they come with increased computational cost.

Over the past decade, advances in computational power and the exponential growth of high-quality flow data have led to the emergence of neural networks and data-driven techniques as a promising approach in wall modeling for turbulent flows. Unlike traditional empirical models, NN-based wall models may capture complex, non-linear interactions within the near-wall region by learning from large datasets generated from fully resolved simulations or experimental data. These models can potentially adapt to a wide range of flow conditions, including those involving separation, reattachment, and strong adverse pressure gradients, which are challenging for conventional models. By leveraging data-driven techniques, deep neural networks offer the potential to significantly improve accuracy in predicting wall shear stress and other key quantities, while also maintaining computational efficiency [30].

However, challenges remain in ensuring the generality and robustness of these models across different flow regimes and configurations. Despite significant research efforts, a standardized methodology for developing these neural network-based wall models has yet to be established, particularly when it comes to the selection of input and output features for the neural networks. Current approaches vary widely, with some models using local flow quantities such as velocity gradients, wall distances or Reynolds stresses, while others incorporate global or integrated flow characteristics. This lack of consensus hinders the development of a universally applicable framework that can handle a broad range of turbulent flow scenarios.

Objectives

The present thesis aims to develop a wall law using deep learning algorithms capable of fully reconstructing all flow quantities in the near-wall region, providing greater accuracy and generality than traditional models. This wanted model is therefore highly flexible, allowing for various applications such as imposing shear stress either at the wall or higher in the computational domain, or modeling the entire near-wall flow, making it ideal for use in methods like the Immersed Boundary Method (IBM).

Given the methodological nature of this work, it has been chosen to develop the proposed wall model for steady body-fitted RANS simulations, enabling rapid, lightweight simulations that facilitate testing of different modeling approaches. The goal is to closely replicate a wall-resolved RANS simulation by replacing the full boundary layer resolution through a data-driven approach.

Representative academic cases have been selected for training and evaluation, offering a range of flow configurations and boundary layer evolutions while ensuring ease of implementation.

Furthermore, the exploratory nature of this study aims to identify the input variables that would enable a more reliable modeling of boundary layer development. The input parameters used in this model are mainly inspired by the classical variables from the self-similar solution of the inner region of the boundary layer, made dimensionless using the wall shear stress.

Structure of the thesis

The structure of this work is organized as it follows:

Chapter 1: State of the art

Chapter 1 provides a thorough overview of important topics related to the study. It begins by detailing the numerical methods used in computational fluid dynamics (CFD), particularly focusing on the applications of the Reynolds Averaged Navier-Stokes (RANS) equations. Next, it introduces the concept of boundary layer (BL), examining traditional modeling techniques and highlighting the increasing influence of Machine Learning (ML) in modeling wall-bounded flows. The chapter wraps up with an introduction to the core principles of Deep Learning (DL) and neural networks (NNs).

Chapter 2: Methods and tools

Chapter 2 presents the key tools employed in this study and elaborates on the methodology for developing the proposed wall model and its numerical integration into the CFD solver. It concludes by discussing the flow configurations analyzed, detailing their geometry, the finite volume discretization used in the computations and the boundary conditions applied during the computation of the steady state solution.

Chapter 3: Data-driven wall models for RANS

Chapter 3 outlines a data-driven methodology for developing RANS wall models for low Mach number aerodynamic simulations. It first details the wall modeling approach, based on a fully-connected neural network trained to approximate the relationship between nondimensional wall distance, pressure gradient, and velocity. It then presents the flow configurations considered for training and testing. The chapter then covers the implementation of the neural network, including the presentation of methods to optimize its architecture, and the training procedure. The final section shows the results obtained with the model on both training and unseen configurations.

This chapter is derived from the article published in Romanelli et al. [92].

Chapter 4: Efficient data-driven wall models for RANS

Chapter 4 addresses key issues identified in the methodology proposed in Chapter 3, with the goal of enhancing the model's accuracy and reducing computational costs. It first introduces a new data-driven approach for wall models in RANS simulations, reformulating the wall laws as a Dirichlet-To-Neumann (DtN) map. Additionally, the accuracy of the wall law is enhanced through improved modeling of turbulent variables and the inclusion of additional input parameters to better characterize the boundary layer state. Then, the flow configurations considered for training and testing are presented. The performances

in terms of accuracy and computational cost are evaluated and compared to reference fully resolved RANS simulations on seen and unseen configurations. Finally, the model's robustness is tested on a completely different configuration, considering the flow around of an airfoil.

The content of this chapter, presented at the ECCOMAS 2024 conference, has been submitted to the journal "Journal of Computational Physics" and is currently under review.

Chapter 5: Current development and future works

In the last chapter, before concluding, the latest developments and outlines potential future research directions are presented. First, it discusses a methodological approach to detect extrapolation conditions for the neural network, enhancing model robustness and providing valuable feedback for troubleshooting. Next, it explores the application of the model within an IBM framework, addressing key challenges and considerations for successful implementation in such simulations. Finally, the chapter examines extending the model's validity to more complex flow configurations by identifying current limitations and proposing necessary actions to enable its application in cases where the current model would otherwise fail.

Chapter 1

State of the art

1.1	Introduction	14
1.2	Numerical methods for computational fluid dynamics	14
1.2.1	Reynolds Averaged Navier-Stokes Equations	14
1.2.2	Spalart-Allmaras turbulence model	16
1.2.3	Finite volume method	17
1.2.3.1	Convective flux discretization	18
1.2.3.1.1	Roe scheme	18
1.2.3.1.2	Third order MUSCL reconstruction	19
1.2.3.2	Viscous flux discretization	20
1.2.3.3	Numerical stencil	20
1.3	Boundary layer	21
1.3.1	Boundary layers	21
1.3.1.1	Types of boundary layer	22
1.3.1.2	Boundary layer separation	23
1.3.2	Turbulent boundary layer	23
1.3.2.1	Turbulent boundary layer equations	23
1.3.2.2	Self-similar solution of the velocity distribution	24
1.3.2.2.1	Inner layer	25
1.3.2.2.2	Outer layer	26
1.3.2.2.3	Overlap layer	26
1.3.2.3	Turbulent boundary layer structure	27
1.4	Wall modeling	28
1.4.1	Standard wall models	29
1.4.1.1	Differential wall models	30
1.4.1.2	Shear stress models	31
1.4.1.2.1	Analytical wall models	31
1.4.1.2.2	Integral wall models	32
1.4.2	Data-driven wall models	33
1.5	Neural Networks	34
1.5.1	Fully Connected Neural Networks	34
1.5.1.1	Activation functions	35
1.5.2	Training a neural network	36
1.5.2.1	Gradient back-propagation	36
1.5.2.2	Update of neural network parameters	39
1.5.2.3	Common gradient-based optimizers for neural networks	39
1.5.2.4	Neural network accuracy and architecture	40
1.5.2.5	Training techniques and issues	41

1.1 Introduction

This chapter offers a comprehensive overview of key areas relevant to the present study. First, it outlines the numerical methods employed in computational fluid dynamics (CFD), with a specific emphasis on applications of the Reynolds Averaged Navier-Stokes (RANS) equations. It then focuses on the boundary layer (BL), discussing conventional modeling approaches and offering an overview on the growing role of Machine Learning (ML) in this context. The chapter concludes with an introduction to some of the fundamental concepts in Deep Learning (DL) and neural networks (NNs).

1.2 Numerical methods for computational fluid dynamics

To introduce the key numerical aspects relevant to this thesis, this section starts from the governing RANS equations (Spalart-Allmaras turbulent model), then introduces the finite volume discretization technique, detailing the essential steps required to iteratively solve the given set of governing equations. Special emphasis is placed on the spatial discretization techniques, as they significantly influence parts of the work that follows.

1.2.1 Reynolds Averaged Navier-Stokes Equations

The compressible RANS equations for a Cartesian reference frame read:

$$\begin{cases} \frac{\partial \rho}{\partial t} + \frac{\partial \rho u_k}{\partial x_k} = 0, \\ \frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_i u_j}{\partial x_j} = -\frac{\partial p}{\partial x_j} \delta_{ij} + \frac{\partial \tau_{ij}}{\partial x_j}, \\ \frac{\partial \rho E}{\partial t} + \frac{\partial \rho E u_j}{\partial x_j} = -\frac{\partial p u_j}{\partial x_j} - \frac{\partial q_j}{\partial x_j} + \frac{\partial \tau_{kj} u_j}{\partial x_j}, \end{cases} \quad (1.1)$$

where ρ is the density, $\mathbf{u} = (u_1, u_2, u_3)$ is the velocity vector, whose component are respectively defined along the axis x_1 , x_2 and x_3 respectively, p is the pressure, $\bar{\boldsymbol{\tau}}$ is the stress tensor, E is the total energy and \mathbf{q} is the heat flux vector. The equations are expressed using the Einstein summation convention along k and j indexes and the Kronecker delta δ_{ij} . The first equation describes the mass conservation, the second one express the momentum conservation equation along the i direction, while the latter is the energy conservation equation.

The system of equations (1.1) is obtained by time-averaging the Navier-Stokes (NS) equations, decomposed through the Reynolds decomposition [87]. The generic flow quantity $\Phi = \bar{\phi} + \phi'$ is thus composed by a mean quantity $\bar{\phi}$ and a fluctuating component ϕ' . Here, the quantity ϕ refers to mean quantity for the sake of brevity.

The stress tensor $\bar{\boldsymbol{\tau}}$ in equations (1.1) consists of two main components: the viscous stress and the Reynolds stress. The viscous stress relates to the molecular viscosity μ , while the Reynolds stress originates from the turbulent fluctuations u'_i in the velocity field and represents the turbulent transport of momentum.

Following the Stokes hypothesis [81], the stress tensor is defined as

$$\tau_{ij} = \underbrace{\mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3} \frac{\partial u_k}{\partial x_k} \delta_{ij} \right)}_{\text{Viscous stress}} - \underbrace{\overline{\rho u'_i u'_j}}_{\text{Reynolds stress}}, \quad (1.2)$$

using again the Einstein summation convention along k index and the Kronecker delta δ_{ij} .

To effectively solve the equations, it is crucial to model these turbulent fluctuations. Turbulence models are used for this purpose, providing predictions of how the Reynolds stresses contribute to the mean flow.

Following the Boussinesq approximation, the stress tensor $\overline{\boldsymbol{\tau}}$ is defined as

$$\tau_{ij} = \underbrace{\mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3} \frac{\partial u_k}{\partial x_k} \delta_{ij} \right)}_{\text{Viscous stress}} + \underbrace{\mu_t \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3} \frac{\partial u_k}{\partial x_k} \delta_{ij} \right)}_{\text{Reynolds stress}}, \quad (1.3)$$

in which μ_t is the turbulent viscosity, defined by a turbulent model and discussed in the following section.

In equation (1.3), the dynamic viscosity is function of the temperature T and it can be estimated by using the Sutherland law:

$$\mu(T) = \mu_0 \left(\frac{T}{T_0} \right)^{\frac{3}{2}} \frac{T_0 + C}{T + C}, \quad (1.4)$$

where the quantifies μ_0 , T_0 and C_0 are constants defined by the fluid. For the air, they are, respectively, $1.711 \cdot 10^{-5} \frac{\text{kg}}{\text{m.s}}$, 273.16 K and 110.4 K . Moreover, it can be useful to introduce the cinematic viscosity of the fluid, defined by

$$\nu = \frac{\mu}{\rho}. \quad (1.5)$$

Considering the air as a perfect gas, the following relationship holds between pressure, density and temperature:

$$p = R\rho T, \quad (1.6)$$

where R is the perfect gas constant, equal to $287 \frac{\text{J}}{\text{kgK}}$ for the air. The total energy of the flow is defined as

$$E = e + \frac{1}{2} u_i u_i, \quad (1.7)$$

using the Einstein summation convention along the i index. Here, e is the internal energy given by

$$e = C_v T, \quad (1.8)$$

where C_v specific heat capacity at constant volume. The heat flux is defined through the Fourier law as below:

$$q_i = \underbrace{\left(-\lambda \frac{\partial T}{\partial x_i} \right)}_{\text{Laminar heat flux}} + \underbrace{\left(-\lambda_t \frac{\partial T}{\partial x_i} \right)}_{\text{Turbulent heat flux}}. \quad (1.9)$$

As for the stress tensor, it is possible to identify the laminar contribution and the turbulent one, given the thermal conductivities

$$\lambda = \frac{\mu C_p}{Pr} \quad \text{and} \quad \lambda_t = \frac{\mu_t C_p}{Pr_t}, \quad (1.10)$$

where C_p is the specific heat capacity at constant pressure, $Pr = 0.72$ and $Pr_t = 0.9$ are respectively the laminar and turbulent Prandtl number for the air.

1.2.2 Spalart-Allmaras turbulence model

Solving the unclosed system of equations in (1.1) requires a turbulence model to estimate the turbulent viscosity in equations (1.3) and (1.10). Over the years, many approaches have been proposed, but the most common turbulence models usually add additional transport equations to the system of RANS equations to model the required turbulent quantities. The present work will focus on the Spalart-Allmaras turbulence model (S.-A.) [102]. Unlike more complex models such as $k - \epsilon$ [55] or $k - \omega$ [124] models which solve for multiple equations, the S.-A. model simplifies the computational effort by focusing on the transport of a single variable, which is a pseudo-viscosity, defined as $\tilde{\nu}$, and linked to the turbulent viscosity by:

$$\mu_t = \rho \tilde{\nu} f_{v1}. \quad (1.11)$$

It is then possible to define the eddy viscosity as:

$$\nu_t = \frac{\mu_t}{\rho}. \quad (1.12)$$

Although more sophisticated turbulence models exist, the SA model remains widely employed as a standard in aerospace applications due to its simplicity and robustness. Various versions of the S.-A. model have been developed, such as S.-A.-neg [3], S.-A.-RC [98] and S.-A.-LRe [103], among others. This work focuses specifically on the compressible formulation of the original S.-A. model, as proposed by Deck [27]. The transport equation for the S.-A. variable $\tilde{\nu}$ is given by:

$$\underbrace{\frac{\partial \rho \tilde{\nu}}{\partial t} + \frac{\partial \rho u_j \tilde{\nu}}{\partial x_j}}_{\text{Production}} - \underbrace{\rho \left[c_{w1} f_w - \frac{c_{b1}}{\kappa^2} f_{t2} \right] \left(\frac{\tilde{\nu}}{d} \right)^2}_{\text{Destruction}} + \underbrace{\frac{1}{\alpha} \left[\frac{\partial}{\partial x_j} \left(\rho (\nu + \tilde{\nu}) \frac{\partial \tilde{\nu}}{\partial x_j} \right) + c_{b2} \frac{\partial \rho \tilde{\nu}}{\partial x_i} \frac{\partial \tilde{\nu}}{\partial x_i} \right]}_{\text{Diffusion}}, \quad (1.13)$$

where d defines the wall distance, and the other quantities are defined as:

$$\begin{aligned} \tilde{S} &= S + \frac{\tilde{\nu}}{\kappa^2 d^2} f_{v2}, \quad S = \sqrt{2 \Omega_{ij} \Omega_{ij}} \\ f_{v2} &= 1 - \frac{\chi}{1 + \chi f_{v1}}, \quad f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3}, \quad \chi = \frac{\rho \tilde{\nu}}{\mu} \\ f_w &= g \left[\frac{1 + c_{w3}^6}{g^6 + c_{w3}^6} \right]^{1/6}, \quad g = r + c_{w2} (r^6 - r), \quad r = \frac{\tilde{\nu}}{\tilde{S} \kappa^2 d^2}, \\ f_{t2} &= c_{t3} \exp(-c_{t4} \chi^2), \end{aligned} \quad (1.14)$$

in which the vorticity Ω is given by

$$\Omega_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i} \right). \quad (1.15)$$

The model constants are:

$$\begin{aligned} c_{b1} = 0.1355, \quad c_{b2} = 0.622, \quad \sigma = \frac{2}{3}, \quad \kappa = 0.41, \\ c_{w1} = \frac{c_{b1}}{\kappa^2} + \frac{1 + c_{b2}}{\sigma}, \quad c_{w2} = 0.3, \quad c_{w3} = 2, \\ c_{v1} = 7.1, \quad c_{t3} = 1.1, \quad c_{t4} = 0.5. \end{aligned} \quad (1.16)$$

1.2.3 Finite volume method

The RANS equations system in (1.1) can be written in compact form as:

$$\frac{\partial \mathbf{Q}}{\partial t} + \nabla \cdot \mathbf{F}_c(\mathbf{Q}) - \nabla \cdot \mathbf{F}_v(\mathbf{Q}) = 0, \quad (1.17)$$

where \mathbf{F}_c and \mathbf{F}_v are respectively the convective and viscous flux vectors, \mathbf{Q} is the conservative variable vector. These quantities are defined as

$$\mathbf{F}_c(\mathbf{Q}) = \begin{pmatrix} \rho u_j \\ \rho u_i u_j + p \\ (\rho E + p) u_j \end{pmatrix}, \quad \mathbf{F}_v(\mathbf{Q}) = \begin{pmatrix} 0 \\ \tau_{ij} \\ \tau_{ij} u_j - q_j \end{pmatrix}, \quad \text{with} \quad \mathbf{Q} = \begin{pmatrix} \rho \\ \rho u_i \\ \rho E \end{pmatrix}. \quad (1.18)$$

Finally, let us also define the primitive variables vector \mathbf{P} as

$$\mathbf{P} = \begin{pmatrix} \rho \\ u_i \\ T \end{pmatrix}. \quad (1.19)$$

The numerical method used to solve equation (1.17) is based on the Finite Volume Method (FVM) [34]. First, a domain $\Omega \subset \mathbb{R}^n$ is considered, where n represents the dimensionality of the domain. This domain Ω is divided into a grid composed of a finite number of control volumes Ω_i , each having boundary edges $\Gamma_{i,k}$. Integrating (1.17) over the volume of a cell Ω_i , the following expression is obtained:

$$\frac{\partial}{\partial t} \int_{\Omega_i} \mathbf{Q} d\Omega + \int_{\Omega_i} \nabla \cdot \mathbf{F}_c(\mathbf{Q}) d\Omega - \int_{\Omega_i} \nabla \cdot \mathbf{F}_v(\mathbf{Q}) d\Omega = 0. \quad (1.20)$$

Using the Green-Ostrogradski theorem, the following integral form is obtained:

$$\frac{\partial}{\partial t} \int_{\Omega} \mathbf{Q} d\Omega + \int_{\partial\Omega} \mathbf{F}_c(\mathbf{Q}) \cdot \mathbf{n} d\Gamma - \int_{\partial\Omega} \mathbf{F}_v(\mathbf{Q}) \cdot \mathbf{n} d\Gamma = 0, \quad (1.21)$$

where $\partial\Omega$ is the boundary of the cell and \mathbf{n} is the outward unit vector normal to the surface. From equations (1.21), the semi-discrete formulation of the RANS equations for an elementary volume Ω_i can be written as

$$\frac{\partial \bar{\mathbf{Q}}_i}{\partial t} + \frac{1}{|\Omega_i|} \left(\sum_{k \in \partial\Omega_i} (\mathcal{F}_c)_{i,k}^\perp |\Gamma_{i,k}| - \sum_{k \in \partial\Omega_i} (\mathcal{F}_v)_{i,k}^\perp |\Gamma_{i,k}| \right) = 0, \quad (1.22)$$

where $|\Omega_i|$ is the volume of the i -th grid cell and $|\Gamma_{i,k}|$ is the surface of the k -th interface belonging to the i -th cell, while $\partial\Omega_i$ is the whole boundary surface of the i -th cell. The FVM considers a constant flow solution inside any elementary volume of the domain. These quantities $\bar{\mathbf{Q}}_i$ in equation (1.22) are defined as

$$\bar{\mathbf{Q}}_i = \frac{1}{|\Omega_i|} \int_{\Omega} \mathbf{Q} d\Omega. \quad (1.23)$$

The quantities $(\mathcal{F}_c)_{i,k}^\perp$ and $(\mathcal{F}_v)_{i,k}^\perp$, in (1.22), respectively denote the convective and the viscous flux through a boundary face $\Gamma_{i,k}$. These are defined as follows

$$(\mathcal{F}_c)_{i,k}^\perp = \frac{1}{|\Gamma_{i,k}|} \int_{\Gamma_{i,k}} \mathbf{F}_c(\mathbf{Q}) \cdot \mathbf{n} d\Gamma \quad \text{and} \quad (\mathcal{F}_v)_{i,k}^\perp = \frac{1}{|\Gamma_{i,k}|} \int_{\Gamma_{i,k}} \mathbf{F}_v(\mathbf{Q}) \cdot \mathbf{n} d\Gamma. \quad (1.24)$$

The next sections focus on the adopted strategy to discretize both the convective and the viscous fluxes.

1.2.3.1 Convective flux discretization

Convective fluxes represent the physical transport of quantities such as mass and momentum within a fluid. However, directly using convective fluxes in numerical simulations can lead to issues like numerical instability and oscillations, particularly in regions with sharp gradients or discontinuities [35]. In order to ensure the stability, accuracy, and convergence of the numerical solution, instead of directly using convective fluxes, numerical fluxes are usually employed. These incorporate advanced computational schemes designed to mitigate these issues and provide a stable and accurate solution, usually by adding necessary dissipation terms into the system [119].

The conventional numerical convective flux is given by:

$$(\mathcal{H}_c)_{i,k}^\perp = \mathcal{H}_c(\mathbf{Q}_{i,k}^L, \mathbf{Q}_{i,k}^R) \cdot \mathbf{n}_{i,k} \quad (1.25)$$

where the function $(\mathcal{H}_c)_{i,k}$ depends on the chosen scheme to approximate the physical flux $(\mathcal{F}_v)_{i,k}$. The conservative variable vectors $\mathbf{Q}_{i,k}^L$ and $\mathbf{Q}_{i,k}^R$ are calculated respectively on the left and on the right side of the k -th interface of i -th cell, which outward normal is denoted by the $\mathbf{n}_{i,k}$.

Multiple numerical flux schemes exist to address various challenges and requirements in simulating fluid flows, designed to handle different flow conditions, such as shocks, rarefactions, and discontinuities, while balancing between accuracy and stability. Roe [91], AUSM+ [69], HLLC [115] are just an example of the most used ones. The Roe's schemes being mainly used in this work, the next section is devoted to briefly introduce it.

1.2.3.1.1 Roe scheme

The Roe's scheme [91] is a numerical scheme to solve hyperbolic partial differential equations, particularly those that arise in the study of compressible fluid flow. This finite volume method is designed to accurately capture shock waves and discontinuities in the flow, leveraging the concept of flux difference splitting (FDS). The Roe's scheme approximates the solution of the Riemann problem by linearizing the Jacobian matrix of the convective fluxes defined as

$$\mathbf{J}_c(\mathbf{Q}) = \frac{\partial \mathbf{F}_c(\mathbf{Q})}{\partial \mathbf{Q}}. \quad (1.26)$$

The Roe's schemes is defined as:

$$(\mathcal{H}_c)_{i,k} = \frac{1}{2} (\mathbf{F}_c(\mathbf{Q}_{i,k}^L) + \mathbf{F}_c(\mathbf{Q}_{i,k}^R)) - \frac{1}{2} \tilde{\mathbf{J}}_c(\mathbf{Q}_{i,k}^L, \mathbf{Q}_{i,k}^R) (\mathbf{Q}_{i,k}^R - \mathbf{Q}_{i,k}^L), \quad (1.27)$$

where $\tilde{\mathbf{J}}_c(\mathbf{Q}_{i,k}^L, \mathbf{Q}_{i,k}^R)$ is the linearized Jacobian matrix computed at the k -th boundary of the i -th cell, using respectively the left and right states of the interface.

The estimation of the left and right states, denoted as $\mathbf{Q}_{i,k}^L$ and $\mathbf{Q}_{i,k}^R$, is essential for determining the discretization order of the scheme. By using the mean state of the i -th cell, $\bar{\mathbf{Q}}_i$, as the left state, and the mean state of the neighboring cell at the k -th interface as the right state, a first-order accurate discretization is obtained for convective schemes. However, employing reconstruction techniques yields more accurate estimates of the conservative variable vector at the interface, enabling higher accuracy schemes. One method for this is the MUSCL reconstruction [117]. In this work, a third-order MUSCL reconstruction is utilized, and the following sections provide a brief overview of it.

1.2.3.1.2 Third order MUSCL reconstruction

To reconstruct the left and right state of the conservative variables at the interface, the primitive variables are interpolated for the left and right state. To simplify the presentation, the particular case of a one-dimensional structured mesh is considered, with the curvilinear direction described by the grid index i . The primitive variables at the i -th cell are denoted as \mathbf{P}_i . The considered mesh is shown in figure 1.1.

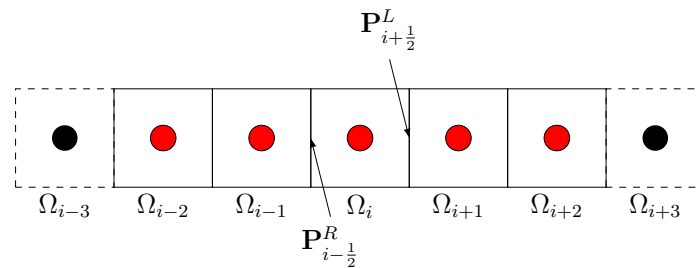


Figure 1.1: Stencil for the third order MUSCL reconstruction. Subpart of a mesh with respect to the direction i . For a cell i , denoted Ω_i , primitive variables \mathbf{P}_i are known at the center of the cell. The values $\mathbf{P}_{i-1/2}^R$ and $\mathbf{P}_{i+1/2}^L$ are computed from five known values \mathbf{P}_{i-2} , \mathbf{P}_{i-1} , \mathbf{P}_i , \mathbf{P}_{i+1} and \mathbf{P}_{i+2} . The stencil of the scheme is represented in red.

The left and right state of primitive variables for the cell interface at $i \pm \frac{1}{2}$ is obtained as follows

$$\begin{cases} \mathbf{P}_{i-1/2}^R &= -\frac{1}{6}\mathbf{P}_{i+1} + \frac{5}{6}\mathbf{P}_i + \frac{1}{3}\mathbf{P}_{i-1} \\ \mathbf{P}_{i+1/2}^L &= \frac{1}{3}\mathbf{P}_{i+1} + \frac{5}{6}\mathbf{P}_i - \frac{1}{6}\mathbf{P}_{i-1}, \end{cases} \quad (1.28)$$

which results in a parabolic reconstruction scheme that is third-order accurate in space.

1.2.3.2 Viscous flux discretization

Viscous fluxes are crucial for accurately representing the behavior of fluids in CFD simulations, affecting shear and heat transfer driven effects in the computed flows. Unlike the convective fluxes, the dissipative nature of viscous ones makes them intrinsically stable, not requiring particular numerical scheme for their computation [42].

In the present work, a second-order centered discretization of the viscous flux is employed, involving two successive operations. The first operation evaluates the velocity gradient, temperature gradient, and potentially the gradient of turbulent quantities. Once these flux densities are determined, the second operation discretizes the diffusion fluxes and calculates their balance.

The gradient evaluation of the physical quantity ϕ at the cell center i along the direction j is obtained from the average cell value using the Green-Ostrogradski theorem :

$$\frac{\partial \phi}{\partial x_j} = \frac{1}{|\Omega_i|} \int_{\Omega} \phi d\Omega = \frac{1}{|\Omega_i|} \int_{\partial\Omega} \phi \cdot \mathbf{n} d\Gamma, \quad (1.29)$$

which can be discretized as it follows

$$\frac{\partial \phi}{\partial x_j} = \frac{1}{|\Omega_i|} \sum_{k \in \partial\Omega_i} \frac{1}{2} |\Gamma_{i,k}| (\phi_i + \phi_{n(i,k)}) \cdot \mathbf{n}_{i,k}, \quad (1.30)$$

where $n_{(i,k)}$ identifies the $\Omega_{n(i,k)}$ elementary cells sharing the k -th interface with the Ω_i cell.

For a bi-dimensional application, the computational stencil for the gradient estimation of a quantity ϕ at the cell center $C_{(i,j)}$ is shown in figure 1.2.

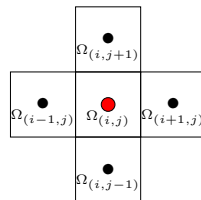


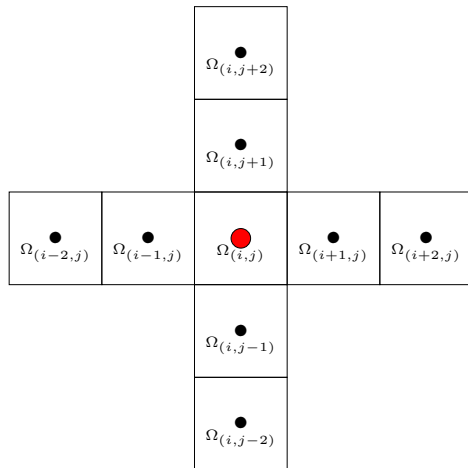
Figure 1.2: Stencil for gradient computation. The gradient at the cell $\Omega_{(i,j)}$ in red is computed using adjacent cells in black.

After the evaluation of the gradient in the first stage, which has just been described, it is then possible to carry out the balance of the diffusion fluxes for the cell Ω_i . As for the calculation of the gradients, each surface integral is evaluated by assuming that the flux density is uniform along the cell boundary $\Gamma_{i,k}$. The viscous flux balance is thus:

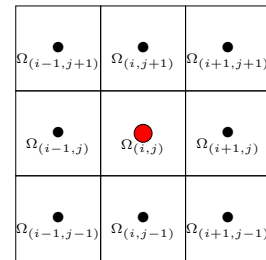
$$(\mathcal{F}_v)_{i,k}^\perp = \frac{1}{2} (\mathbf{F}_v(\bar{\mathbf{Q}}_i) + \mathbf{F}_v(\bar{\mathbf{Q}}_{n(i,k)})) \cdot \mathbf{n}_{i,k}. \quad (1.31)$$

1.2.3.3 Numerical stencil

This work presents bi-dimensional applications of the above discretized method. It is thus important to define the numerical stencil useful for a given bi-dimensional cell $\Omega_{(i,j)}$ for the convective and numerical flux. Considering a third order MUSCL reconstruction and a second-order centered discretization of the viscous flux, the involved cells are respectively shown in figures 1.3a and 1.3b.



(a) Numerical stencil for convective flux computation using third order MUSCL reconstruction



(b) Numerical stencil for viscous flux computation

Figure 1.3: Numerical stencil for convective and viscous flux computation in a 2D case. The fluxes for the cell $\Omega_{(i,j)}$ in red are computed using adjacent cells in black. The numerical stencil for the cell $\Omega_{(i,j)}$ is given by the superposition of convective and viscous stencils.

1.3 Boundary layer

This section aims to introduce the fundamentals of boundary layer (BL) theory, providing a comprehensive exploration of its formation, characteristics, and behavior, with a particular focus on turbulent boundary layers.

1.3.1 Boundary layers

As a fluid flows past a stationary object, the molecules in direct contact with the object surface adhere to it due to the fluid viscosity. The molecules just above this layer are decelerated by collisions with these stationary molecules, exchanging their momentum with them. These slowed molecules, in turn, reduce the speed of the molecules in the layers above them. As the distance from the surface increases, the influence of these collisions diminishes, and the momentum exchange decreases. This creates a thin layer of fluid near the surface, where the velocity transitions from zero at the surface to the free stream value farther away. This thin layer of fluid is known as the boundary layer (BL) [88].

From a continuum perspective, molecules can be considered as grouped into fluid layers. When molecules exchange momentum, a force is generated between two layers of fluid, representing the rate of change of momentum. This force per unit area is known as shear stress and is directly proportional to the velocity gradient [97], and it is defined as

$$\tau = \mu \frac{\partial u_{\parallel}}{\partial y}, \quad (1.32)$$

where u_{\parallel} is the tangential velocity with respect to the wall surface. The thickness of the boundary layer (BL) increases along the downstream direction as shown in figure 1.4, which sketches the velocity distribution in the BL over a flat plate. In front of the leading edge, the velocity distribution is uniform. With increasing distance from the leading edge in the downstream direction, the thickness δ of the retarded layer increases continuously, as increasing quantities of fluid become affected.

One common way to define the boundary layer thickness δ is the distance to the wall where the flow velocity inside the boundary layer reaches 99% of the external velocity magnitude or asymptotic velocity, i.e. $U(\delta) = 0.99 U_e$.

1.3.1.1 Types of boundary layer

As firstly shown by Reynolds [90], boundary layers (BLs) can be categorized into two primary types: laminar and turbulent. The laminar BL is characterized by smooth, orderly fluid flow with minimal mixing. However, under certain conditions, this laminar layer can transition to a turbulent one, where the fluid exhibits chaotic flow patterns with significant mixing and a sudden and large increase in the BL thickness, as illustrated in figure 1.4. The transition process is complex and remains not fully understood today. However, three main categories of turbulent transition have been identified. Natural transition occurs at relatively low levels of free-stream turbulence and is driven by the growth of small disturbances within the laminar flow. These disturbances amplify and eventually lead to turbulence, due to viscous effects, eventually breaking down into smaller vortices that form turbulent spots [97]. Bypass transition, as described by Morkovin [77], is triggered by large disturbances in the external flow [123] and, lastly, forced transition involves the deliberate introduction of disturbances, such as tripping devices like rough patches or surface protrusions [36].

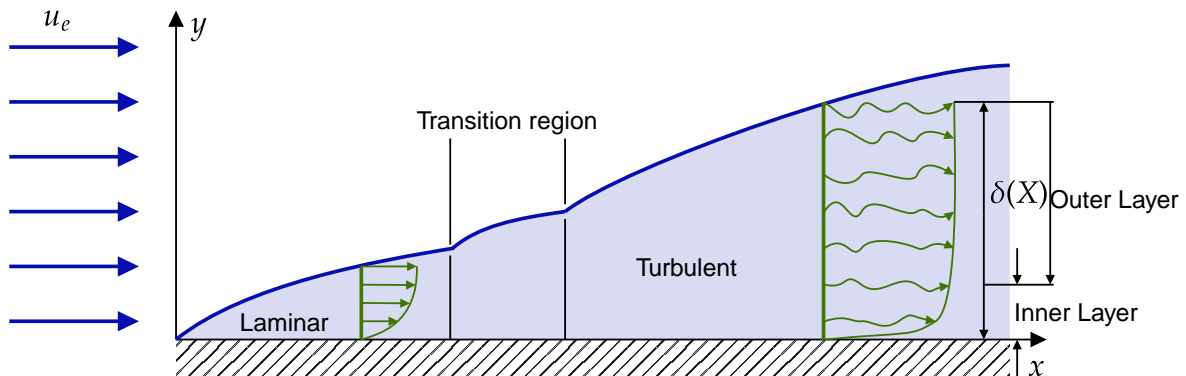


Figure 1.4: Representative scheme of the boundary layer structure

In the case of natural transition, the nature of the boundary layer (BL) can be determined through the Reynolds number [90], firstly introduced by Stokes [108]

$$Re = \frac{\mu u_\infty L}{\rho} = \frac{u_\infty L}{\nu}, \quad (1.33)$$

in which L represents a characteristic length of the flow, typically selected based on convention. For example, the diameter is often used to describe flow around spheres or circles. In the case of flow within a pipe, the internal diameter is commonly adopted, while for flows around objects, either the length or width may be chosen.

The Reynolds number provides a criterion for determining the nature of the flow, whether laminar or turbulent. It essentially quantifies the ratio of inertial forces to viscous forces within the fluid. Laminar flow, characterized by smooth and orderly movement, typically occurs at low Reynolds numbers where viscous forces are predominant. In contrast, turbulent flow, which is marked by chaotic eddies, vortices, and flow instabilities, arises at high Reynolds numbers, where inertial forces dominate [44].

The Reynolds number can be expressed locally, Re_x being thus the local Reynolds number and it is defined as

$$Re_x = \frac{\mu u_\infty x}{\rho}, \quad (1.34)$$

in which x defines the streamwise location where the Reynolds number is computed. In case of a flat plate flow, showing a natural transition from laminar boundary layer to turbulent boundary layer, the local Reynolds number can be used to identify the transition location, which arises at $Re_x \approx 3 - 5 \cdot 10^5$ [100].

1.3.1.2 Boundary layer separation

The decelerated fluid particles in the BL do not, in all cases, remain in the thin layer which adheres to the body along the whole wetted length of the wall. In some cases, especially when a strong adverse pressure gradient is present in the upstream direction, the decelerated particles are forced outwards. This causes the boundary layer (BL) to separate from the wall. This phenomenon is described as separation and it is always associated with the formation of vortices and a sudden thickening of the decelerated layer of particles. The point of separation is classically identified by the following condition:

$$\left. \frac{\partial u_{\parallel}}{\partial y} \right|_{y=0} = 0. \quad (1.35)$$

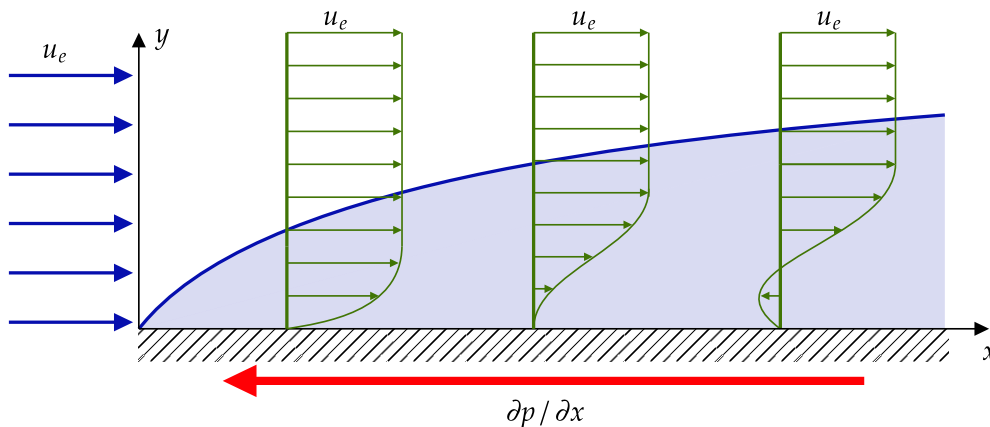


Figure 1.5: Representative scheme of the boundary separation under adverse pressure gradient

1.3.2 Turbulent boundary layer

This section highlights the key characteristics of the turbulent boundary layer (TBL). First, the governing equations describing its behavior are presented. These are then simplified to derive the self-similar solution for the velocity evolution within the boundary layer. Finally, the structure of the TBL is discussed.

1.3.2.1 Turbulent boundary layer equations

First, the mass conservation and momentum equations from the NS equations are considered, focusing on the bi-dimensional incompressible case for simplicity. The equations

are expressed in a reference frame where the x -direction is aligned with the wall surface, and y represents the Cartesian coordinate in the wall-normal direction. For the velocity vector $\mathbf{u} = (u_1, u_2) = (u_{\parallel}, u_{\perp})$, the velocity component tangent to the wall is defined as u_{\parallel} , while u_{\perp} represents the normal component to the wall. As for the RANS equations, both mean and turbulent flow quantities are accounted for. A generic flow variable $\Phi = \bar{\phi} + \phi'$ consists of the sum of a mean $\bar{\phi}$ and a fluctuating component ϕ' . By time-averaging the NS equations, the following set of equations is obtained

$$\begin{cases} \frac{\partial \bar{u}_i}{\partial x_i} = 0 \\ \rho \left(\bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} \right) = -\frac{\partial \bar{p}}{\partial x_i} + \mu \frac{\partial^2 \bar{u}_i}{\partial x_j^2} - \rho \frac{\partial \overline{u'_i u'_j}}{\partial x_j}. \end{cases} \quad (1.36)$$

This set of equations is then simplified using the boundary layer approximation, which assumes that $\bar{u}_{\perp} \ll |\mathbf{u}|$ and $\frac{\partial}{\partial x} \ll \frac{\partial}{\partial y}$, leading to the following simplified form:

$$\begin{cases} \frac{\partial u_{\parallel}}{\partial x} + \frac{\partial u_{\perp}}{\partial y} = 0 \\ \rho \left(u_{\parallel} \frac{\partial u_{\parallel}}{\partial x} + u_{\perp} \frac{\partial u_{\parallel}}{\partial y} \right) \approx -\frac{\partial p_e}{\partial x} + \mu \frac{\partial^2 u_{\parallel}}{\partial y^2} - \rho \frac{\partial \overline{u'_{\parallel} u'_{\perp}}}{\partial y} \\ 0 \approx -\frac{\partial p}{\partial y} - \rho \frac{\partial \overline{u'^2_{\perp}}}{\partial y}, \end{cases} \quad (1.37)$$

in which p_e is the pressure outside of the BL. The simplification below

$$p_e \approx p + \rho \overline{u'^2_{\perp}} \quad (1.38)$$

is in fact given by the TBL momentum equation in the wall normal direction.

In the following, the overline $\bar{\cdot}$ is omitted and any quantity ϕ refers to mean quantity for the sake of brevity.

1.3.2.2 Self-similar solution of the velocity distribution

From equations (1.37), assuming a simple shear stress flow with no external pressure gradients, such as in a flat-plate boundary layer or a fully-developed pipe or channel flow, the momentum equation in the streamwise direction x simplify to

$$0 = \mu \frac{\partial^2 u_{\parallel}}{\partial y^2} - \rho \frac{\partial \overline{u'_{\parallel} u'_{\perp}}}{\partial y} \quad (1.39)$$

It results that in this configuration the shear stress remains constant in the wall normal direction

$$\tau = \underbrace{\mu \frac{\partial u_{\parallel}}{\partial y}}_{\tau_v} - \underbrace{\rho \overline{u'_{\parallel} u'_{\perp}}}_{\tau_t} = \frac{\partial u_{\parallel}}{\partial y} \Big|_{y=0} = \tau_w = \text{constant}. \quad (1.40)$$

There are two different mechanisms by which the momentum component can be carried over through the flow: the molecular momentum transfer due to the viscosity τ_v and the momentum transfer due to turbulent fluctuations τ_t .

Very close to the wall, the most important scaling parameters are the kinematic viscosity ν and wall shear stress τ_w . This allows to introduce the characteristics velocity and length scales for near wall flows:

$$u_\tau = \sqrt{\frac{\tau_w}{\rho}} \quad \text{and} \quad \delta_v = \frac{\nu}{u_\tau}, \quad (1.41)$$

respectively defined as friction velocity and viscous length scale [87]. Additionally, using the velocity and length scales introduced above, it is possible to define the friction Reynolds number

$$Re_\tau = \frac{u_\tau \delta}{\nu} = \frac{\delta}{\delta_v}. \quad (1.42)$$

Using the flow scaling parameters, the velocity and the wall distance can be expressed in wall units as

$$u^+ = \frac{u_\parallel}{u_\tau} \quad \text{and} \quad y^+ = \frac{y}{\delta_v} = \frac{u_\tau y}{\nu}, \quad (1.43)$$

where u^+ is the dimensionless wall velocity and y^+ is the dimensionless wall distance and represents a measure of the relative importance of viscous and turbulent transport at different distances from the wall.

Moreover, the distance from the wall can be scaled using the boundary layer thickness δ as

$$\eta = \frac{y}{\delta}. \quad (1.44)$$

Adimensioning equation (1.40) through the characteristics scales of the flow, the equations

$$\frac{1}{Re_\tau} \frac{\partial u^+}{\partial \eta} + \tau_t^+ = 1 \quad (1.45)$$

is obtained, in which $\tau_t^+ = \frac{\tau_t}{\rho u_\tau^2}$. For large Reynolds numbers, $Re_\tau \rightarrow \infty$, the momentum transfer due to molecular viscosity becomes negligible in comparison to the turbulent momentum transfer. This approximation holds well throughout most of the boundary layer but breaks down near the wall, where velocity fluctuations diminish to zero. At the wall, in fact, τ_t^+ vanishes. It is thus clear that for large Reynolds numbers, two main zones can be identified in the boundary layer structure. The first far from the wall, where viscous effects are negligible. Since this condition holds over most of the boundary layer, the thickness of this layer is of the order of magnitude of δ . The second region is closer to the wall, where τ_t^+ approaches zero. Here, both molecular momentum transfer through viscosity and turbulent momentum transfer are significant. The thickness of this viscous layer is in the order of $\delta_v = \frac{\delta}{Re_\tau}$, making it much smaller than the overall boundary layer thickness δ and independent of it at large Reynolds numbers.

Since fully-developed boundary-layer flow is completely specified by the six parameters u_\parallel , y , ρ , ν , δ and u_τ on the three independent dimensions, dimensional analysis and the Pi-theorem [16] yield a functional relationship between three dimensionless groups, conveniently taken as

$$\frac{u_\parallel}{u_\tau} = f\left(\frac{y}{\delta_v}, \frac{y}{\delta}\right) = f(y^+, \eta). \quad (1.46)$$

1.3.2.2.1 Inner layer

The near wall viscous layer, as previously stated, present a length scale much smaller than the entire boundary layer thickness, thus resulting independent of it. For this reason it is

useful to scale the equation (1.40) through near wall scaling parameters, u_τ and δ_v , which yields

$$\frac{\partial u^+}{\partial y^+} + \tau_t^+ = 1. \quad (1.47)$$

Equation (1.47) then reduces to

$$u^+ = f_w(y^+). \quad (1.48)$$

Moreover, knowing that at the wall $\tau_t^+ = 0$, in proximity of the wall equation (1.48) simplifies to

$$u^+ \approx y^+. \quad (1.49)$$

Since the inner layer is independent of the outer layer, the solution is expected to be universal or self-similar, meaning that, through appropriate scaling, the velocity profiles for different boundary layers converge to this common solution.

1.3.2.2.2 Outer layer

As introduced before, in the outer layer, the effects of viscosity are negligible for high Reynolds number. For this reason, a solution of the velocity is searched in the form of

$$u^+ = f_o(\eta). \quad (1.50)$$

Since the outer layer is independent from near wall viscous flow, the main influence here is found in the external flow of the boundary layer. The velocity profile evolution is thus often described as a velocity-defect law, with reference to outer flow velocity u_e . The form of the velocity evolution is

$$\frac{u_e - u_{\parallel}}{u_\tau} = \frac{u_e^+ - u^+}{u_\tau} = f_o(\eta). \quad (1.51)$$

1.3.2.2.3 Overlap layer

The inner layer and the outer layer, driven respectively by viscous and turbulent momentum transfer, meet in an overlap zone in which turbulent and viscous effect coexist. Looking for a solution for the velocity evolution $u^+ = f_w(y^+, \eta)$, one may enforce a smooth overlap between inner and outer layer velocity profiles. This is obtained by summing equation (1.48) and (1.51):

$$u_e^+(Re_\tau) = f_o(\eta) + f_w(y^+). \quad (1.52)$$

where $y^+ = \eta Re_\tau$, leading to the expression

$$u_e^+(Re_\tau) = f_o(\eta) + f_w(\eta Re_\tau). \quad (1.53)$$

Differentiating with reference to Re_τ , it gives

$$u_e'^+(Re_\tau) = 0 + \eta f_w'(\eta Re_\tau), \quad (1.54)$$

and further differentiating with respect to η yields:

$$0 = f_w'(\eta Re_\tau) + \eta Re_\tau f_w''(\eta Re_\tau), \quad (1.55)$$

which can be rewritten as:

$$0 = f'_w(y^+) + y^+ f''_w(y^+) \quad (1.56)$$

$$= \frac{\partial}{\partial y^+} \left(y^+ \frac{\partial f_w(y^+)}{\partial y^+} \right). \quad (1.57)$$

Hence,

$$y^+ \frac{\partial f_w(y^+)}{\partial y^+} = \text{constant}, \quad (1.58)$$

in which the constant is conventionally written as $\frac{1}{\kappa}$, where $\kappa \approx 0.41$ is Von Kármán's constant:

$$\frac{\partial f_w(y^+)}{\partial y^+} = \frac{1}{\kappa y^+}. \quad (1.59)$$

As noted by Millikan [74], the inner and outer layers can only overlap smoothly if the overlap region velocity profile is logarithmic. Integrating equation (1.58), the following expression is obtained

$$f_w(y^+) = \frac{1}{\kappa} \ln(y^+) + C^+, \quad (1.60)$$

in which the constant $C^+ \approx 5$ is experimentally determined.

As for the viscous layer, this solution is expected to be a universal self-similar solution, generally referred to as law of the wall.

1.3.2.3 Turbulent boundary layer structure

The two layer structure of the TBL sees, as stated before, an outer layer, in which the viscous effects are negligible and an inner layer, closer to the wall where the momentum transfers is a combination of viscous effects and turbulent fluctuation. These two layers meet on an overlap region, being respectively the upper and lower boundary for inner and outer layers.

However, a finer description of the TBL structure exists. A turbulent boundary layer (TBL) can be divided into four regions: the viscous sublayer, the buffer layer, the logarithmic layer or overlap region and the wake region [87].

The inner layer, encompassing the viscous sublayer, buffer layer, and logarithmic layer ends at approximately $\frac{y}{\delta} \approx 0.15$, where δ represents the boundary layer (BL) thickness [97]. In the viscous sublayer, the viscous shear stress is dominant, overshadowing the turbulent shear stress, as turbulent fluctuations are suppressed in this thin layer close to the wall [58]. Conversely, in the logarithmic layer, turbulent shear stress predominates, allowing the viscous shear stress to be neglected [87]. Between these two layers lies the buffer layer, where viscous and turbulent shear stresses are of comparable magnitude, making both significant [97]. Pope [87] gives the following delimitation of the zones and self-similar solution applicability:

$$\begin{cases} u^+ = y^+ & y^+ \leq 5 & \text{(Viscous sublayer)} \\ & 5 < y^+ \leq 30 & \text{(Buffer layer)} \\ u^+ = \frac{1}{\kappa} \log(y^+) + C^+ & y^+ > 30 & \text{(Logarithmic layer)} \end{cases} \quad (1.61)$$

The wake region, belonging to the outer layer, also known as the velocity-defect layer, is where the velocity profile deviates from the logarithmic law, a deviation that becomes more pronounced in non-equilibrium BL with adverse pressure gradients [22].

A schematic representation of the evolution of the velocity profile and the shear stress along the TBL structure layers are given in figures 1.6 and 1.7.

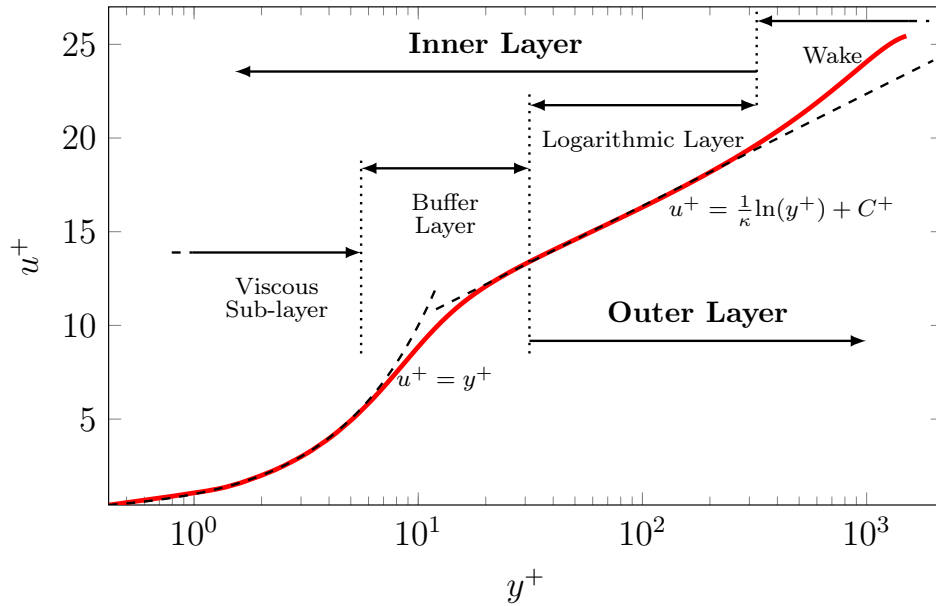


Figure 1.6: Scheme of the turbulent boundary layer structure

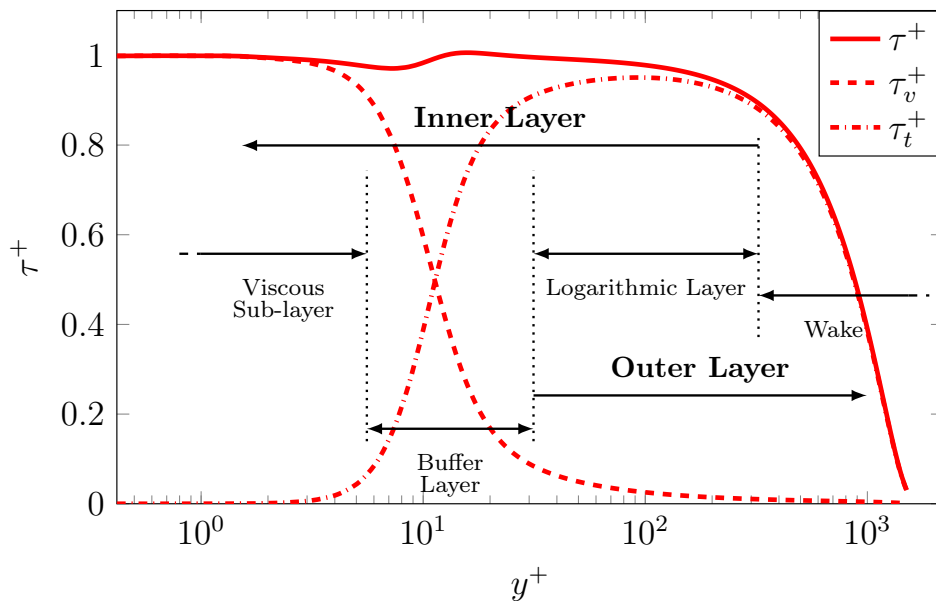


Figure 1.7: Shear stress evolution in a turbulent boundary layer.

1.4 Wall modeling

This section introduces the concept of wall modeling. It starts by explaining the general reasoning for using a wall model in CFD simulations, building on the boundary layer concepts covered in previous sections. Then, it provides an overview of different wall modeling approaches found in the literature, along with their main applications.

The main purpose of wall modeling, especially for turbulent boundary layer flows, is to avoid solving the complete set of boundary layer equations near the wall, where turbulence scales become extremely small, and flow state gradients, particularly in the wall-normal direction, are very steep, as previously discussed. In high Reynolds number flows, this near-wall region is characterized by complex interactions between viscous forces and turbulence, and resolving it directly would require an extremely fine mesh, increasing computational costs [86]. A schematic example of this is shown in figure 1.8.



Figure 1.8: Representation of boundary layer (BL) capturing with different mesh refinements

Furthermore, since $\frac{\partial}{\partial x} \ll \frac{\partial}{\partial y}$ in the boundary layer, accurately capturing the near-wall gradients requires a high aspect ratio in the mesh discretization, which negatively affects the overall mesh quality [112].

Moreover, in the case of Immersed Boundary Method (IBM), the necessity of wall models is even more pronounced. Since IBM does not explicitly represent the wall within the domain discretization, with the mesh not being body-fitted. This means that there is not any direct resolution of the physical wall. Instead, the interaction between the flow and the boundary is imposed through a forcing term [75]. As a result, accurately resolving the near-wall behavior is particularly challenging, and a wall model becomes necessary to reproduce the effects of the wall.

The wall laws allow to solve the mentioned issues by replacing the direct solution of the boundary layer flow with a model that approximates the behavior of the flow close to the wall, while the flow far from the wall is solved [86]. Generally, the wall model needs input data from the flow field at a certain height above the wall. This height, called the matching point or sampling point [57], is a critical interface between the resolved flow (further from the wall) and the modeled flow (closer to the wall) [128]. At this sampling point, the flow state is extracted and these quantities are used to drive the wall model, providing the necessary boundary conditions for the resolved flow in the near-wall region, by modeling the shear stress, the velocity evolution and, usually, the thermodynamic state of the flow [62]. This data exchange allows the solution to maintain consistency between the near-wall model and the fully resolved outer flow [9].

This approach allows complex flows to be simulated with reduced computational effort, making it suitable for industrial applications such as aerodynamic surfaces, turbines, and heat exchangers [86, 62].

1.4.1 Standard wall models

In the literature, different approaches to wall modeling are proposed. As Larsson et al. [62] suggest, the wall model methodology can mainly be split in two categories: the differential methods and the wall-stress models.

1.4.1.1 Differential wall models

In differential methods, the resolved region of the flow is separated from the approximated region by an interface. This approach allows for reduced grid refinement in the computationally expensive near-wall region by either modeling or partially resolving the fine-scale turbulent structures and steep velocity gradients through less resource-intensive techniques. The key advantage of differential methods is their ability to lower computational costs by relaxing fine grids requirements, while still accurately capturing the essential physics of turbulent boundary layers. This makes them especially useful for complex flows, featuring separations, reattachment or shock-wave boundary layer interactions, which are often challenging for pure modeling techniques [105].

A representative scheme of the application of a differential wall model is given in figure 1.9.

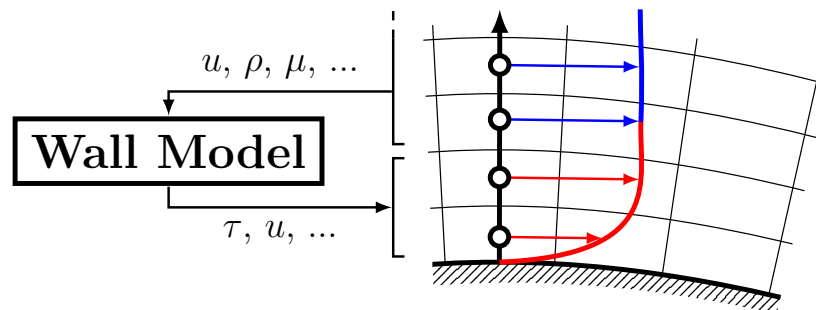


Figure 1.9: Representative scheme of a differential wall model (WM) application

This approach is widely adopted for Large Eddy Simulations (LESs), which are coupled with RANS equations in Hybrid RANS/LES approaches. The core concept is to adopt the RANS turbulence models to solve the flow state where the grid resolution is not fine enough for LES approaches to capture turbulent structures. This allows the method to transition between RANS, which models the averaged behavior of the flow, and LES, which resolves larger turbulent eddies, depending on the local grid density and flow characteristics.

Hybrid RANS/LES approaches can be categorized into two main types: zonal methods and seamless methods [11]. In zonal methods, the interface between the RANS and LES regions is predefined at a certain distance from the wall. The LES computation is performed in the region above this interface, where it provides Dirichlet boundary conditions for the RANS calculations below the interface. In return, the RANS region supplies Neumann boundary conditions to the coarse LES grid above the interface [83]. In seamless methods, the RANS/LES interface is not fixed but depends both on the grid and on the solution. It is the case of the approach Detached Eddy Simulation (DES), specifically designed to address high Reynolds number, massively separated flows. In DES, a RANS closure is used for boundary layers while the massively separated regions are computed with LES [104].

Therefore, it is evident that differential wall models tend to be complex and costly to implement, as they require solving systems of ordinary or partial differential equations in space [9]. This set of equations are generally obtained through boundary layer simplification of NS equations, as in (1.37), adopting RANS closure terms to model the Reynolds stress tensor [6].

1.4.1.2 Shear stress models

Unlike differential approaches, wall shear stress models extend the resolved region up to the wall surface, where the simple no-slip condition is replaced by the enforcement of wall shear stress. Since mesh refinement near the wall is relaxed, applying the standard no-slip boundary condition becomes impractical. As previously mentioned, coarse grids lead to inaccuracies in estimating the velocity gradient, as shown in figure 1.8b, which affects the prediction of the velocity profile and the growth of the boundary layer. To address this, the wall model must provide appropriate boundary conditions for the wall shear stress. This shear stress is determined at the wall by utilizing information from the outer flow and properly modeling the inner flow region [11].

A representative scheme of the application of a shear stress wall model is given in figure 1.10.

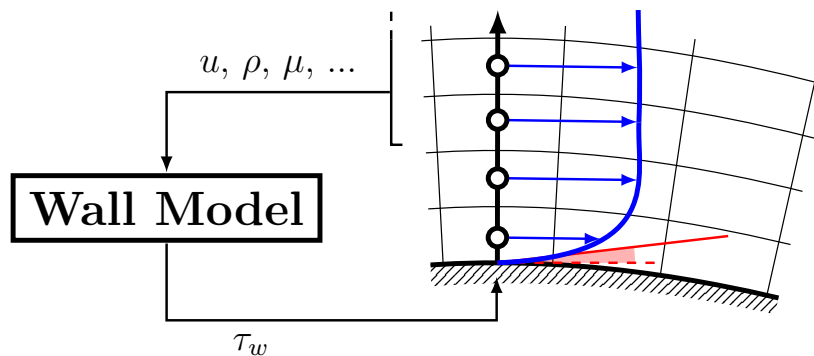


Figure 1.10: Representative scheme of a shear stress wall model (WM) application

Shear stress models are generally less computationally demanding than differential models, as they do not require solving ODEs or PDEs in space. The two main types of wall models are analytical and integral wall models, both heavily reliant on the self-similar solution of the turbulent boundary layer inner layer, discussed in section 1.3.2.2.

1.4.1.2.1 Analytical wall models

The analytical models allow for the direct calculation of wall shear stress based on the velocity at a specific wall-normal position, referred to as the sampling point. The wall shear stress is determined by identifying the value that satisfies the modeled velocity profile at this point. Due to the two-layer structure of the self-similar solution of the TBL, a smoothing or blending procedure between the viscous sublayer and the logarithmic region is necessary to ensure the law's validity across the entire inner boundary layer. Several authors have proposed blending approaches between the linear and logarithmic layers, such as Spalding [106] and Musker [78], who introduced the following

$$u^+ = 5.424 \tan^{-1} \left[\frac{2y^+ - 8.15}{16.7} \right] + \log_{10} \left[\frac{(y^+ + 10.6)^{9.6}}{(y^{+2} - 8.15y^+ + 86)^2} \right] - 3.52. \quad (1.62)$$

Additionally, wall models can be derived from the analytical solution of RANS turbulence closure models by making certain simplifying assumptions about the flow configuration. For example, the Spalart-Allmaras wall law [110] establishes a relationship between the dimensionless wall velocity u^+ and the dimensionless wall distance y^+ , defined as follows:

$$u^+(y^+) = \bar{B} + c_1 \log \left[(y^+ + a_1)^2 + b_1^2 \right] - c_2 \log \left[(y^+ + a_2)^2 + b_2^2 \right] - c_3 \arctan \left[y^+ + a_1, b_1 \right] - c_4 \arctan \left[y^+ + a_2, b_2 \right], \quad (1.63)$$

where the coefficients are

$$\begin{aligned} \bar{B} &= 5.03339088, & a_1 &= 8.14822158, & a_2 &= -6.92870938, \\ b_1 &= 7.46008761, & b_2 &= 7.46814579, \\ c_1 &= 2.54967735, & c_2 &= 1.33016516, \\ c_3 &= 3.59945911, & c_4 &= 3.63975319. \end{aligned} \quad (1.64)$$

This wall model is derived from the analytical solution of the Spalart-Allmaras turbulence model, applied to a zero-pressure gradient (ZPG) boundary layer developing over a flat plate.

Moreover, some wall models aim to overcome the limitation of the self-similar solution, which is restricted to ZPG conditions, by adapting existing models to account for streamwise pressure gradients. This adaptation introduces a new non-dimensional parameter, the streamwise wall pressure gradient p^+ . These laws are thus defined as

$$u^+ = f(y^+, p^+), \quad \text{with} \quad p^+ = \frac{\mu_w}{\rho_w^2 u_\tau^3} \frac{\partial p}{\partial x}. \quad (1.65)$$

The streamwise wall pressure gradient p^+ has been introduced by Afzal [1], who gave an analytical expression for the evolution of the dimensionless streamwise velocity

$$u^+ = \kappa^{-1} \left[\log(y^+) - 2 \log \left(\frac{\sqrt{1 + p^+ y^+}}{2} \right) + 2 \left(\sqrt{1 + p^+ y^+} - 1 \right) + \kappa C^+ \right]. \quad (1.66)$$

Although Afzal's wall law incorporates the effects of pressure gradients on boundary layer evolution, equation (1.66) is only valid in the logarithmic region of the boundary layer and it is applicable exclusively for adverse (i.e., positive) pressure gradients due to the square root in the expression. In practice, this restriction may lead to implementation difficulties and potential convergence issues when applying the wall model.

1.4.1.2.2 Integral wall models

On the other hand, integral wall models are derived from the simplification of RANS equations with the boundary layer hypothesis. Especially from the momentum equation along the streamwise direction and integrated from the wall up to a sampling point far from the wall. The wall shear stress is then obtained as a function of the external pressure gradient

$$\tau_w = (\mu + \mu_t) \frac{\partial u_{\parallel}}{\partial y} \Big|_{y=y^S} - \frac{\partial p_e}{\partial x} y^S - \frac{\partial L_x}{\partial x} + \rho u_{\parallel} u_{\perp} \Big|_{y=y^S}, \quad (1.67)$$

where y^S defines the wall distance of the sampling point and $L_x = \int_0^{y^S} \rho u_{\parallel}^2 dy$ is unknown and it is usually defined by approximating the inner layer velocity evolution through analytical expressions [126].

1.4.2 Data-driven wall models

Over the past decade, the fluid mechanics community has progressively adopted Machine Learning (ML) and Deep Learning (DL) techniques for various applications, including dimensionality reduction, turbulence modeling, flow control, uncertainty quantification, and optimization [31, 14, 120, 82, 131].

With recent advances in computational power and the exponential accumulation of high-quality data, the use of deep neural networks in wall modeling has emerged as a promising approach. Notably, data-driven wall models have been particularly developed and applied in the context of Large Eddy Simulation simulations.

One of the earliest efforts to apply supervised learning to wall modeling is found in Yang et al. [127]. The authors observed that a model trained on turbulent channel flow data at a single Reynolds number could be extrapolated to higher Reynolds numbers within the same configuration. Huang, Yang, and Kunz [51] focused on spanwise rotating turbulent channels, comparing a fully data-driven approach with an enhanced physics-based method. They improved the neural networks by reorganizing the dimensionless input and output variables, simplifying the functional relationship between them.

These wall shear stress models target the channel flow configurations and attempt to equal or outperform the standard wall models. More recent studies have focused on overcoming the limitations of conventional wall models, particularly in handling strong adverse pressure gradients and flow separation, by employing ML and DL techniques.

Zhou, He, and Yang [131] addresses turbulent separation by training a feed-forward neural network using DNS data from various periodic hill geometries. In this study, a wall law is developed using deep learning approaches, distinguishing itself from previous works by not relying on the traditional friction-based scaling of model inputs and outputs. While the model produced satisfactory a priori results, the a posteriori tests revealed discrepancies between the predictions and the reference case for the periodic hill case.

Lozano-Durán and Bae [70] proposed that complex flows can be represented as a nonlinear combination of simpler building-block flows. Based on this idea, they developed a wall-flux-based model using a self-critical machine learning approach, successfully training it on DNS data from various flow cases (e.g., flat plate, channel, turbulent duct, and separated flows at different Reynolds numbers). However, when applied to the NASA Juncture Flow [96], the model failed to accurately predict the flow separation. Better results are obtained in a later work by combining the wall modeling with the Sub-Grid Scale (SGS) model for LES [4].

Zhou et al. [132] trained a neural network mixing DNS data from a two-dimensional periodic hill and synthetic data to predict the two wall-parallel components of wall shear stress. The synthetic data, derived from the law of the wall, covers a wide range of friction Reynolds numbers and wall-normal heights. The a priori tests demonstrated a clear improvement in wall shear stress predictions when using the combined dataset, compared to training solely on periodic hill data.

Another approach in machine learning is Reinforcement Learning (RL). Bae and Koumoutsakos [5] applied RL, originally designed for flow control, to develop a wall model for ZPG turbulent boundary layers, specifically in channels and flat plates. In this approach, the neural network is trained within the simulation framework directly. Inputs and outputs are non-dimensionalized using viscosity and the modeled friction velocity from the previous time step, with the reward function based on the presence of a logarithmic layer near the wall. Reinforcement Learning has also been utilized to create new wall models for separated flows, as shown in Zhou et al. [130]. Rather than directly predicting

wall shear stress, which the authors found problematic due to inconsistencies between the actual near-wall flow direction and the flow velocity at distant sampling points, this model focuses on predicting eddy viscosity as a boundary condition.

Recent literature has shown growing interest in data-driven wall modeling, particularly in the context of wall shear stress modeling. These models mainly seek to address the limitations of traditional approaches by leveraging neural networks to improve accuracy and handle complex flow phenomena. Although various ML and data selection strategies have been explored, a standardized approach for defining model inputs and outputs is still lacking. Nevertheless, the flexibility of NNs in modeling intricate and nonlinear behaviors presents immense potential for advancing wall flow simulations. This potential for improved precision and wider applicability suggests a promising path for future research and development in wall modeling.

1.5 Neural Networks

Nowadays, neural networks are pivotal in a wide range of scientific and industrial domains, including image analysis [60], data mining [45], natural language processing [118], robotics and autonomous vehicle control [67], cybersecurity [68], chemistry and drug discovery [66], finance [54], and physics [17]. Their ability to process complex data and recognize patterns has led to breakthroughs in these fields, revolutionizing how tasks such as automated image classification, predictive analytics and language translation are approached.

The development of neural networks begins with the conceptual foundation laid by McCulloch and Pitts in the 1940s, who introduced a mathematical model of a neuron [73]. This early work inspired subsequent research, leading to the creation of the perceptron by Rosenblatt in the 1950s, a basic type of neural network capable of learning binary classifications [93]. The 1980s saw the advent of Convolutional Neural Networks (CNNs) by LeCun, which were designed to process and analyze spatial data such as images [64]. Meanwhile, Recurrent Neural Networks (RNNs), developed to handle sequential data, were enhanced in the 1990s by Hochreiter and Schmidhuber with Long Short-Term Memory Networks (LSTMs), addressing limitations in retaining long-term dependencies [50]. Autoencoders, introduced for data compression and feature learning [49] and Transformers, which revolutionized natural language processing with self-attention mechanisms [118], further advanced the field. These innovations collectively paved the way for the widespread and impactful use of neural networks (NNs) in various modern applications.

1.5.1 Fully Connected Neural Networks

Even if multiple architectures of neural networks have been proposed, regression problems see a widespread application of Fully Connected Neural Networks (FCNNs). This kind of NN, also called Multilayer Perceptron (MLP), is adopted through this work. This section focuses on introducing their main characteristics about architecture and training.

A FCNN consists of a series of dense node layers, with each of the nodes fully connected with the ones on the previous and following layer. Every node of a single layer receives information from all nodes belonging to the previous layer and passes information to all nodes of the next layer. Since the node layers cascade into one another, the FCNNs are classified as a Feedforward Neural Networks (FNNs), meaning that the flow of information is uni-directional along the neural network.

In a FCNN, the first layer is defined as input layer and its dimension, i.e. the number of

nodes of which it is composed, matches the dimension of the input space. The dimension of the last layer, called output layer, is constrained by the dimension of the output space. Input and output layer are separated by multiple hidden layers, which are not constrained in number and dimension.

Since Fully Connected Neural Network (FCNN) is the only type of neural network employed in the current work, by default, the NN abbreviation will refer to a FCNN in the following.

Each layer l of the N_L in the NN, each composed by $n^{(l)}$ neurons, transforms a given input vector $\mathbf{I}^{(l)} \in R^{n^{(l)}}$ into an output vector $\mathbf{O}^{(l)} \in R^{n^{(l+1)}}$. The superscript $(\cdot)^{(l)}$ denotes quantities associated with the l -th layer, while $(\cdot)^{(l\pm 1)}$ referring respectively to the previous and the following layer. Each layer thus performs the following transformation

$$\mathbf{O}^{(l)} = L^{(l)} \left(\mathbf{O}^{(l-1)} \right) \quad \forall l \in \{0, \dots, N_L - 1\}. \quad (1.68)$$

Starting from the input vector $\mathbf{I} = \mathbf{I}^{(0)}$, the output $\mathbf{O} = \mathbf{O}^{(N_L-1)}$ of the FCNN is computed in a forward pass of the network as:

$$\mathbf{O} = L^{(N_L-1)} \circ L^{(N_L-2)} \dots L^{(1)} \circ L^{(0)} (\mathbf{I}), \quad (1.69)$$

approximating a continuous function $g : R^{n^{(0)}} \rightarrow R^{n^{(N_L-1)}}$.

Considering individually any single neuron, the output $O_i^{(l)}$ of the i -th neuron is given by:

$$O_i^{(l)} = f_i^{(l)}(W_i^{(l)} + b_i^{(l)}), \quad (1.70)$$

where $f_i^{(l)}$ is the so-called activation function of the layer (more information are given in the following), $b_i^{(l)}$ is a scalar value, called also bias, and $W_i^{(l)}$ is the weighted inputs to the node i of layer l , obtained from a linear combination of the outputs of the previous layers:

$$W_i^{(l)} = \sum_{j=1}^{n^{(l-1)}} w_{i,j}^{(l)} O_j^{(l-1)}, \quad (1.71)$$

where $w_{i,j}^{(l)}$ is the weight coefficient linking the j -th neuron from layer $l-1$ to the i -th neuron of layer l . The weights coefficients and biases are the trainable scalar parameters that are optimized during the training phase of a neural network.

1.5.1.1 Activation functions

The activation functions add non-linearity to the network, allowing it to fit complex non-linear patterns. This enables the NN to be trained on more complex tasks and perform better than a simple linear regression on data. Cybenko [26] proved that a two-layer neural network is a universal function approximator. Moreover, activation functions can highly influence the training of the neural network. More information is given in the following.

In the literature, multiples types of activation function exist. The most common activation functions can be divided into three categories: ridge functions, radial functions and fold functions. However, for FCNN, the most common ones are the ridge activation functions. Examples are Rectified Linear Unit (ReLU) [79], Exponential Linear Unit (ELU) [20], Leaky Rectified Linear Unit (Leaky ReLU) [71], Softplus [40], the hyperbolic tangent or Sigmoid. They are shown in table 1.1.

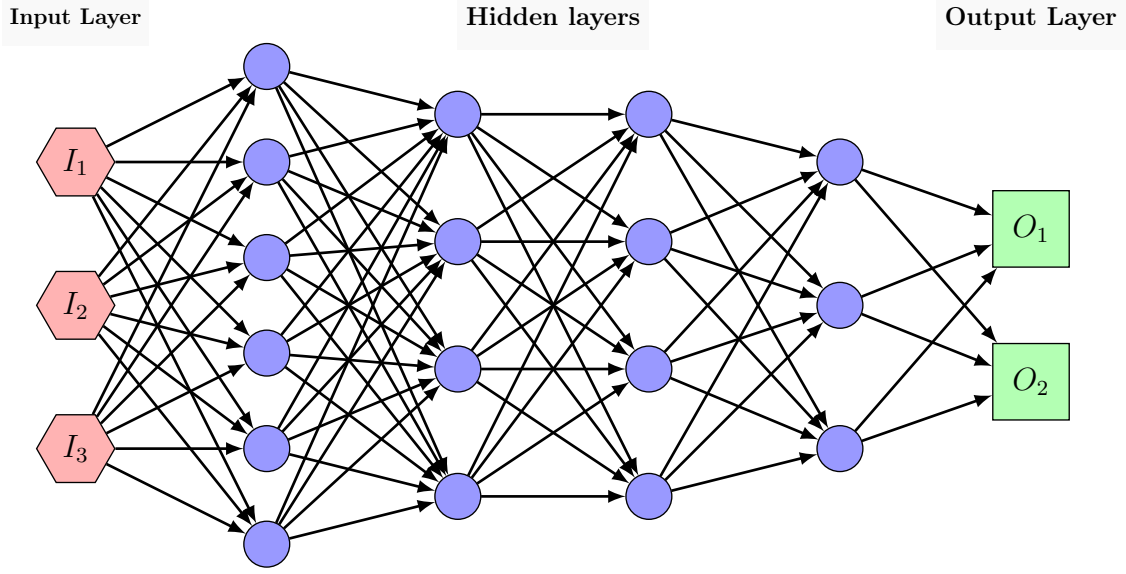


Figure 1.11: Schematic representation of a Fully Connected Neural Network (FCNN)

1.5.2 Training a neural network

The accuracy of a neural network approximating the relation $\mathbf{O} = g(\mathbf{I})$ is generally quantified by a scalar metric of the approximation error between the ground truth $\hat{\mathbf{O}}$ and the estimated quantity \mathbf{O} by the NN. This metric is called as loss or cost function and it is defined in the following by ϵ . The most common loss function are Mean Squared Error (MSE), Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE), defined respectively as

$$\epsilon_{\text{MSE}} = \frac{1}{N_S} \sum_{i=1}^{N_S} (\mathbf{O}_i - \hat{\mathbf{O}}_i)^2, \quad (1.72)$$

$$\epsilon_{\text{MAE}} = \frac{1}{N_S} \sum_{i=1}^{N_S} |\mathbf{O}_i - \hat{\mathbf{O}}_i|, \quad (1.73)$$

and

$$\epsilon_{\text{MAPE}} = \frac{1}{N_S} \sum_{i=1}^{N_S} \left| \frac{\mathbf{O}_i - \hat{\mathbf{O}}_i}{\mathbf{O}_i} \right| \times 100, \quad (1.74)$$

in which N_S indicates the number of samples in the dataset.

Training a NN means modifying its parameters $\boldsymbol{\theta} = \{w_{i,j}^{(l)}, b_i^{(l)}\}$ in order to minimize the loss function over a given data-set. This procedure of comparing the estimation of the NN and to a training dataset is defined as supervised learning [76]. The parameter set $\boldsymbol{\theta}^*$ which minimize the loss function are thus given by:

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \left\| \epsilon(\hat{\mathbf{O}}, \mathbf{O}) \right\|. \quad (1.75)$$

1.5.2.1 Gradient back-propagation

Equation (1.75) is generally iteratively solved through a gradient descent optimization process [41]. This requires the gradient computation of the loss function ϵ with respect to each of the parameter $\boldsymbol{\theta}$, written as

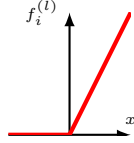
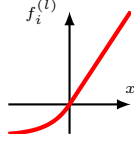
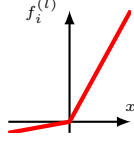
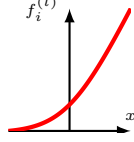
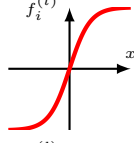
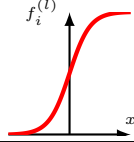
Activation Function	$f_i^{(l)}(x)$	Visualization
ReLU	$f_i^{(l)}(x) = \max(0, x)$	
ELU	$f_i^{(l)}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$	
Leaky ReLU	$f_i^{(l)}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$	
Softplus	$f_i^{(l)}(x) = \log(1 + e^x)$	
Hyperbolic tangent	$f_i^{(l)}(x) = \tanh(x)$	
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	

Table 1.1: Most common activation functions for Fully Connected Neural Network (FCNN)

$$\nabla_{\theta} \epsilon = \left(\frac{\partial \epsilon}{\partial w_{i,j}^{(l)}}, \dots, \frac{\partial \epsilon}{\partial b_i^{(l)}}, \dots \right). \quad (1.76)$$

Backpropagation [15] by automatic differentiation provides an efficient way to compute the gradient of a loss function by applying the chain rule to neural networks, working backward from the final layer.

From equation (1.76), the gradient of ϵ with respect to a weight $w_{i,j}^{(l)}$ is computed applying the chain rule as

$$\begin{aligned}
\frac{\partial \epsilon}{\partial w_{i,j}^{(l)}} &= \frac{\partial \epsilon}{\partial O_i^{(l)}} \frac{\partial O_i^{(l)}}{\partial w_{i,j}^{(l)}} \\
&= \frac{\partial \epsilon}{\partial O_i^{(l)}} \frac{\partial}{\partial w_{i,j}^{(l)}} f_i^{(l)} \left(\sum_{j=1}^{n^{(l-1)}} w_{i,j}^{(l)} O_j^{(l-1)} + b_i^{(l)} \right) \\
&= \frac{\partial \epsilon}{\partial O_i^{(l)}} f_i^{\prime(l)} \left(\sum_{j=1}^{n^{(l-1)}} w_{i,j}^{(l)} O_j^{(l-1)} + b_i^{(l)} \right) O_j^{(l-1)}.
\end{aligned} \tag{1.77}$$

Introducing the activation $a_i^{(l)}$ of the i -th node of the layer l

$$a_i^{(l)} = \sum_{j=1}^{n^{(l-1)}} w_{i,j}^{(l)} O_j^{(l-1)} + b_i^{(l)}, \tag{1.78}$$

and the sensitivity $\delta_i^{(l)}$ of the cost function to the i -th neuron output $O_i^{(l)}$ of the layer l

$$\delta_i^{(l)} = \frac{\partial \epsilon}{\partial O_i^{(l)}}, \tag{1.79}$$

it is possible to obtain the gradient of ϵ with respect to a weight $w_{i,j}^{(l)}$

$$\frac{\partial \epsilon}{\partial w_{i,j}^{(l)}} = \delta_i^{(l)} f_i^{\prime(l)} \left(a_i^{(l)} \right) O_j^{(l-1)}. \tag{1.80}$$

Similarly, the gradient of ϵ with respect to a bias $b_i^{(l)}$ is computed, giving

$$\frac{\partial \epsilon}{\partial b_i^{(l)}} = \delta_i^{(l)} f_i^{\prime(l)} \left(a_i^{(l)} \right). \tag{1.81}$$

The sensitivity $\delta_i^{(l)}$ is simply computed as the derivative of the loss function ϵ with respect to the output $O_i^{(N_L-1)}$ of the NN for the output layer $N_L - 1$:

$$\delta_i^{(N_L-1)} = \frac{\partial \epsilon}{\partial O_i^{(N_L-1)}}. \tag{1.82}$$

On the other hand, for the hidden layers, the chain rule can be applied again

$$\begin{aligned}
\delta_i^{(l)} &= \frac{\partial \epsilon}{\partial O_i^{(l)}} = \sum_{k=1}^{n^{(l+1)}} \frac{\partial \epsilon}{\partial O_k^{(l+1)}} \frac{\partial O_k^{(l+1)}}{\partial O_i^{(l)}} \\
&= \sum_{k=1}^{n^{(l+1)}} \delta_k^{(l+1)} \frac{\partial O_k^{(l+1)}}{\partial O_i^{(l)}} \\
&= \sum_{k=1}^{n^{(l+1)}} \delta_k^{(l+1)} \frac{\partial}{\partial O_i^{(l)}} f_k^{(l+1)} \left(\sum_{i=1}^{n^{(l)}} w_{k,i}^{(l+1)} O_i^{(l)} + b_k^{(l+1)} \right) \\
&= \sum_{k=1}^{n^{(l+1)}} \delta_k^{(l+1)} f_k^{\prime(l+1)} \left(a_k^{(l+1)} \right) w_{k,i}^{(l+1)},
\end{aligned} \tag{1.83}$$

considering the sensitivity of the loss function ϵ with respect to the output $O_k^{(l+1)}$ of the k -th node of the following layer $l + 1$, as well as the sensitivity of the output $O_k^{(l+1)}$ of the following layer nodes with respect to $O_i^{(l)}$.

Once the forward pass of the neural network is completed for a given input \mathbf{I} , knowing the activations $a_i^{(l)}$ and the outputs $O_i^{(l)}$ of every neuron in the NN, it is possible to compute the sensitivities $\delta_i^{(l)}$ in a backward pass, from the output layer to the input layer, in order to compute the gradient $\nabla_{\theta} \|\epsilon\|$. Here, the sensitivity $\delta_i^{(N_L-1)}$ of the output layer is generally computed to algorithmic differentiation of the loss ϵ .

1.5.2.2 Update of neural network parameters

Once the gradient of the loss ϵ with respect to the NN parameters is computed, the latter are updated iteratively following

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \alpha \nabla_{\boldsymbol{\theta}} \|\epsilon\|, \quad (1.84)$$

where t indicates the current iteration, commonly called epoch and α is an update parameter defined as learning rate, which can be both constant or adaptively computed by an optimization algorithm to improve convergence speed and accuracy. A brief overview of the most common optimization algorithm is given in the following.

As clear from equations (1.80) and (1.81), SGD algorithms through backpropagation needs to compute the first order derivative of the activation function $f_i^{(l)}$. Being continuous differentiable for an activation function is thus an advantageous property for gradient-based optimization methods. Even, if optimization is still possible for non continuously differentiable activation functions, it is the case of the ReLU for instance, presents some challenges for optimization [101]. Moreover, some activation functions, which present a saturation

$$\lim_{x \rightarrow \pm\infty} f_i^{(l)}(x) = 0, \quad (1.85)$$

like in the case of hyperbolic tangent, present the problem of vanishing gradients [61]. This leads to a vanishingly small derivatives, preventing parameter update and thus possibly freezing training.

1.5.2.3 Common gradient-based optimizers for neural networks

Since the NN embodies a non-linear function and the gradient descent optimization is a local optimization algorithm [25], the training process could lead to convergence to local minima of the loss function ϵ . Moreover, the computation of the gradient $\nabla_{\boldsymbol{\theta}} \|\epsilon\|$ along all the training dataset can be computationally expensive. In order to tackle these problems, a common practice consists in computing the loss and the relative gradient $\nabla_{\boldsymbol{\theta}} \|\epsilon\|$ only for a reduced portion of the training dataset \mathcal{T} . Any of this portion $\mathcal{B} \subset \mathcal{T}$ is defined as batch.

During a training epoch, the training dataset \mathcal{T} is thus divided in smaller subsets and the update of parameters $\boldsymbol{\theta}$ takes places after the evaluation of the loss function over each of the considered batches, multiple times for a single epoch.

Here, the gradients $\nabla_{\boldsymbol{\theta}} \|\epsilon\|$, computed only on a portion of the training dataset, provide an approximation of the real update directions for parameters $\boldsymbol{\theta}$ given by the totality of \mathcal{T} . However, this cheaper estimation reveals more robust to local minima convergence, avoiding it or allowing to escape from them. This concept is the foundation of the Stochastic Gradient Descent (SGD) algorithms [10]. However, SGD often suffers from noisy updates

and may struggle with convergence, especially when the constant learning rate is not properly tuned. To address these issues, Momentum optimizers enhance the SGD algorithm by incorporating a memory of past gradients. This helps smooth out updates, accelerates convergence, and reduces oscillations, resulting in a more consistent descent path [109]. Further amelioration is provided by the selection of an adaptive learning rate specific for each parameter. It is the case of Root Mean Square Propagation (RMSprop) algorithm. The idea is to adapt the per-parameter learning rate by considering a running average of the recent update magnitudes for that parameter [48]. Lastly, Adaptive Moment Estimation (Adam) algorithm, which will be widely employed in the following of this work, is one of the most popular optimizers due to its robustness and adaptability. It effectively integrates the advantages of both Momentum and RMSprop by calculating an adaptive learning rate for each parameter [89]. The updates are derived from a running average of both the per-parameter gradient and its second moment, determined through an L2-norm of $\nabla_{\theta} \|\epsilon\|$. This approach enables larger updates in regions with high gradient without necessitating fine-tuning of the learning rate.

Even training a basic Fully Connected Neural Network involves a considerable number of parameters that control various aspects of the model's learning process and performances, i.e. accuracy and computational cost. The next sections focus the main aspects to consider when configure and train a NN.

1.5.2.4 Neural network accuracy and architecture

The first aspect to consider is the architecture of the neural network. While the input and output layers are primarily determined by the dimensions of their respective spaces, the design of the hidden layers remains flexible and can be freely customized. The number of hidden layers and their widths, i.e. the number of neurons composing them, are typically adjusted through experimentation and observation, rather than being derived from theoretical principles.

Generally, deeper and wider networks are capable of capturing more complex patterns, but they also demand more data and computational resources, especially during inference. The size of the training dataset significantly influences the choice of architecture: smaller datasets often necessitate the use of more compact neural networks, whereas larger networks may be required to handle more extensive datasets.

Moreover, smaller networks tend to learn more slowly than larger ones [82]. Therefore, when inference costs are not a significant constraint, it is common to choose oversized neural networks, regardless of the expected complexity of the function to be fitted, as long as there is enough data to train them effectively. An example of the training of different sizes NNs is given in figure 1.12.

An effective way to reduce the size of a NN and/or its computational cost is pruning [12]. Pruning can be performed in either a structured or unstructured manner. Structured pruning involves removing the least influential neurons from the network based on the norm of their output $O_i^{(l)}$, which effectively reduces the network's size and the amount of memory required for a forward pass. Unstructured pruning, on the other hand, involves setting the least influential weights and biases to zero. Although this does not directly reduce the size of the network, since the total number of neurons and operations required for a forward pass remain unchanged, unstructured pruning can still reduce computational load during inference through the use of sparse matrix computation. A schematic representation of pruning is given in figure 1.13.

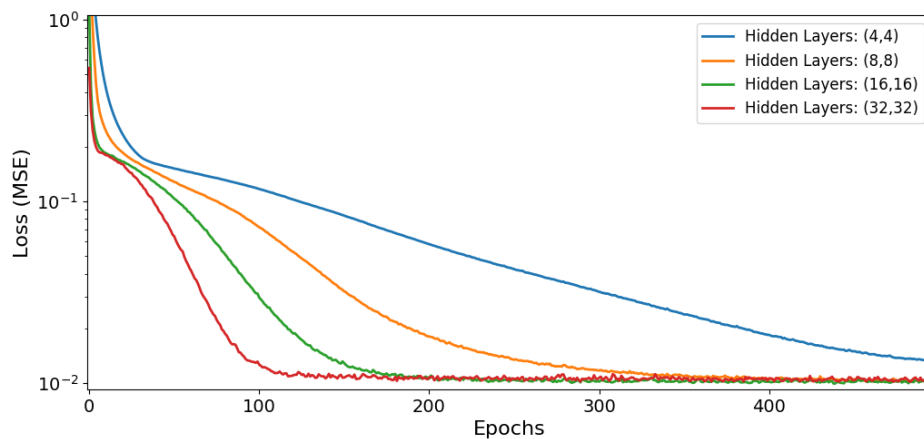


Figure 1.12: Example of accuracy performance over training for different size neural network

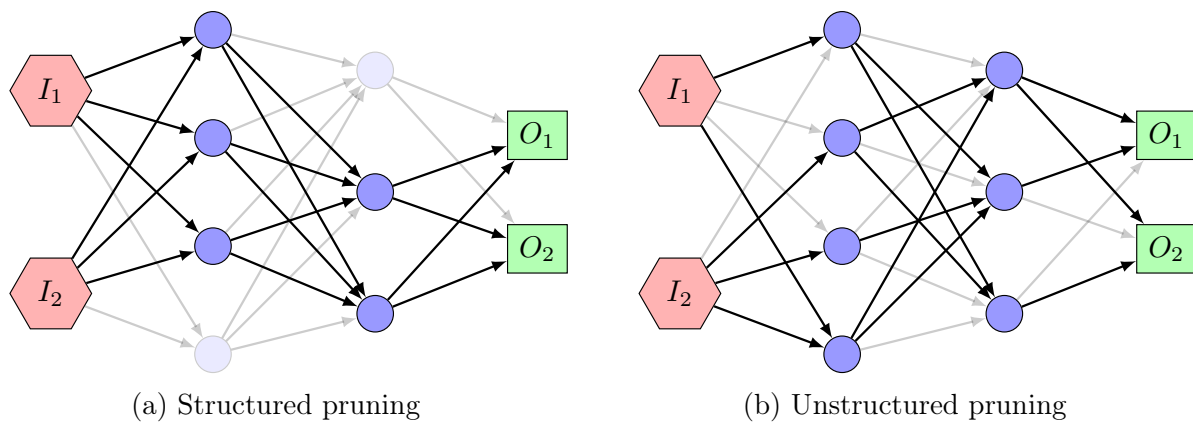


Figure 1.13: Schematic representation of pruning strategy on neural network (NN)

1.5.2.5 Training techniques and issues

Unlike model parameters, which are learned during training, hyperparameters must be set prior to the start of training. They are usually tuned using empirical methods, often relying on previous experience and experimental testing. This process involves trying out different hyperparameter combinations, evaluating the model's performance, and selecting the set of values that produces the best results.

This section outlines the common training techniques employed to set these hyperparameters effectively, as well as the main training related issues.

The first aspect to consider during the training is the initialization of the parameters θ . Using a uniform initialization scheme leads to poor performance because, when an input is propagated through the network, the output of hidden neurons in each layer will have identical effects on the cost function. This results in identical gradients for all neurons within the layer. Consequently, all neurons update in the same manner during training, which hinders their ability to learn different and distinct features. Consequently, an uneven initialization is required, this is generally accomplished through a random initialization. However, attention has still to be paid in order to avoid the first output values from exploding, due to a too high values initialization or from leading to a low convergence rate due to small values initialization and vanishing gradients. A common practice is to

choice random initialization, ensuring that the mean of the activations $a_i^{(l)}$ is zero and that their variance across every layer is constant. It is the case of the He [47] and Xavier Glorot [39] initialization, both of them especially designed to cope with the non-linearity of specific activation functions, respectively for rectifiers, i.e. ELU, ReLU or Leaky ReLU, and sigmoid.

A common issue in neural network training through supervised learning is represented by over-fitting [113], which means that the neural network not only learns the main patterns of the input-output relation but also the noise and biases of the training dataset. As a result, the network performs exceptionally on the training data but fails to generalize to new, unseen data, leading to poor performance through application cases. This phenomenon is more likely to appear when employing complex models, which consist in a large number of parameters θ , thus presenting a high capacity to memorize the training data rather than generalizing main patterns from them. The main causes of over-fitting can be identified in a training on a relatively small dataset or if the training is led for too many epochs.

To detect and prevent overfitting, a common approach is to allocate a portion of the dataset for validation during training. This involves training the model on one part of the dataset (the training set) while using the other part (the validation set) to assess the model performance after each epoch. Overfitting is identified when the model's accuracy on the validation set begins to decline, indicating that the model is memorizing the training data rather than generalizing to new data. Thus, training should be stopped as soon a validation accuracy deterioration or an increase in validation loss is observed.

Another common training strategy consists in adding a penalty or regularization to the training loss ϵ'

$$\epsilon' = \epsilon + \frac{\lambda}{N_S} \|\boldsymbol{\theta}^{(l)}\|, \quad (1.86)$$

in which λ is a hyperparameter that controls the weight of the penalization during training, N_S represents the number of samples in the training dataset and, finally, $\|\boldsymbol{\theta}^{(l)}\|$ denotes the norm of the parameters in a given layer l . The norm is typically the absolute value norm or the quadratic norm, corresponding to L_1 or L_2 penalization, respectively. Penalization acts by discouraging overly complex models, limiting predominant weights and biases in the NN. An example of detecting and mitigating overfitting through regularization is provided in figure 1.14.

Moreover, overfitting can be mitigated by using the Dropout strategy during training. This technique involves randomly deactivating a fraction of the neurons in the neural network during each training epoch. While this may slightly prolong the training process, the goal is to prevent the model from relying too heavily on any particular pathway of information flow through the network, thereby encouraging the development of more robust and generalized representations.

All these methods and techniques rely on hyperparameters that are generally tuned empirically.

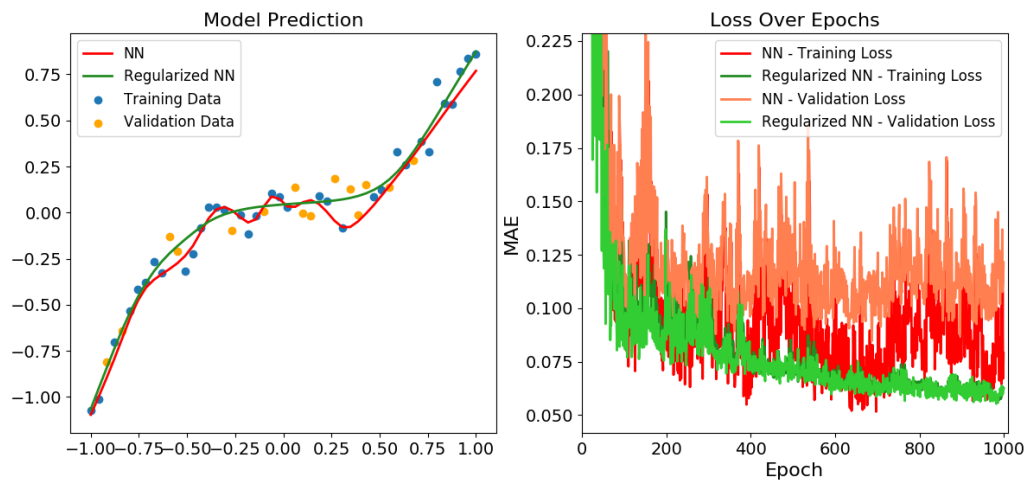


Figure 1.14: Example of overfitting. Comparison between reference neural network (NN) and same NN trained with L_2 regularization.

Chapter 2

Methods and tools

2.1	Methodology and tools for data-driven wall models	45
2.1.1	Workflow of the data-driven wall model	46
2.1.2	Machine Learning Framework	46
2.1.2.1	Data Preprocessing	47
2.1.2.1.1	Non-dimensionalization	47
2.1.2.1.2	Normalization	47
2.1.2.1.3	Transformation	48
2.1.2.2	Neural network training	48
2.1.2.2.1	Neural network architecture	49
2.1.2.2.2	Loss function	49
2.1.2.2.3	Sample weighting	50
2.1.2.2.4	Optimizer and training strategy	51
2.1.3	CFD solver	53
2.1.3.1	Numerical approach	53
2.1.3.2	Neural network integration	53
2.2	Wall model strategy	54
2.2.1	Wall model strategy in a finite volume framework	54
2.2.2	Ghost cells approach	55
2.2.3	Model components	56
2.3	Main test cases	56
2.3.1	2D Bump case	56
2.3.1.1	Domain discretization	58
2.3.2	Airfoil case	58
2.3.2.1	Domain discretization	59

This chapter introduces the primary tools used in this study and details the methodology for developing the proposed wall model and its numerical integration into the CFD solver. Finally, it presents the flow configurations considered, including their geometry, the finite volume discretization employed in the computations and the boundary conditions.

2.1 Methodology and tools for data-driven wall models

This section outlines the methodology required to develop and utilize a data-driven wall model. It provides an overview of each step involved and highlights the tools used for proposing and applying these models. Particular attention is then given to the most critical aspects of this methodology.

2.1.1 Workflow of the data-driven wall model

The process of developing and applying a data-driven wall model in a CFD solver involves two main phases: the offline phase and the online phase. The offline phase focuses on building and training the model, while the online phase deals with its real-time application.

The offline phase begins with constructing the dataset. In this work, the goal is to closely replicate a wall-resolved RANS simulation. This is achieved by deriving the training dataset from a series of RANS simulations that cover a variety of flow configurations. Using a CFD solver, these simulations are conducted to generate the necessary data. Further details about the dataset and the corresponding training dataset are provided in sections 3.3 and 4.3, which are addressed to the specific formulation of the data-driven wall model outlined in their respective chapters. The evolution of the inner region of the boundary layer is then extracted to create data tuples that characterize the boundary layer across different flow conditions.

This data is then fed into a machine learning framework. Initially, the data undergoes preprocessing to enable effective training of the neural network and ensure it accurately captures the dynamics of wall-bounded flows. The preprocessing involves scaling, normalizing, and weighting the importance of each sample. Further details are provided in section 2.1.2.1. After preparing the features, a suitable machine learning model is selected and trained on the data, as described in section 2.1.2.2. Once trained, the model is integrated into the CFD solver by saving its hyperparameters, architecture, and parameters, which are then transferred to the solver for use in the online phase. The integration of neural networks in the solver is described in section 2.1.3.2.

In the online phase, the neural network is loaded into the CFD solver to leverage its capabilities during computations. This integration occurs during the simulation warm-up phase, when the case configuration, including mesh and boundary conditions are provided to the solver. The model configuration is then stored in the solver memory, ready for application. During the computation, the model is used to enforce boundary conditions along the wall surface.

A schematic representation of the workflow for building and applying a data-driven wall model is illustrated in Figure 2.1.

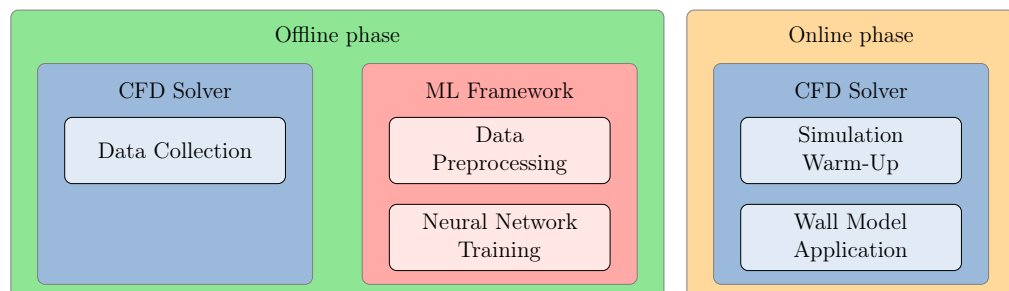


Figure 2.1: Workflow of the data-driven wall model

2.1.2 Machine Learning Framework

Neural networks and machine learning tasks are handled through the TensorFlow library [28], which, like other frameworks, provides automatic differentiation for seamless back-propagation of training gradients and includes a broad range of mathematical

operators, such as activation functions or optimizers. Models are built as computational graphs and persistence functions allow for the serialization and saving of both model architecture and parameters. To facilitate the integration of neural networks into CFD workflows, the TensorFlow saving process has been modified to generate a custom configuration file. This modification organizes NN parameters and architectural details in a readable and structured format, simplifying the integration of the neural network into CFD code.

The following of this section details the essential steps for data preprocessing and model training, including data preparation, selecting the model architecture, configuring hyperparameters, and establishing the training loop with back-propagation and optimization techniques.

2.1.2.1 Data Preprocessing

As introduced in section 2.1.1, a series of CFD simulations across various boundary layer configurations are used to generate the training dataset for the regression NN. However, before being fed to the neural network for the training process, this raw data undergoes preprocessing to enhance the learning capabilities and improve the overall model performance.

2.1.2.1.1 Non-dimensionalization

To ensure the consistency of the functions learned by the regression models, each input and output of the neural networks is non-dimensionalized. Whenever applicable, classical wall scaling based on wall shear stress is employed. Further details regarding this approach and the model formulation are provided in the subsequent chapters.

2.1.2.1.2 Normalization

The non-dimensionalized input feature are normalized. Normalization of the input data of neural network is a crucial preprocessing step that improves model performance and training efficiency. Neural networks, particularly those using gradient-based optimization, are sensitive to the scale of input features. When input data vary significantly in magnitude, the training process can become unstable, leading to slow convergence or poor results.

Normalization typically involves scaling the data so that the dataset range or its statistical properties, such as the standard deviation, are adjusted to a specific scale. Most used techniques are standardization, in which each input features is scaled so that their mean is zero and their standard deviation is unitary, which ensures that the data are centered and scaled consistently [65]. Alternatively, min-max normalization transforms the input features to lie within a specific range, often between 0 and 1 [7]. The min-max normalization and the standardization are defined respectively as

$$I'_n = \frac{I_n - I_{n\text{MIN}}}{I_{n\text{MAX}} - I_{n\text{MIN}}} \quad \text{and} \quad I'_n = \frac{I_n - \bar{I}_n}{\sigma_n}, \quad (2.1)$$

in which $I_{n\text{MIN}}$, $I_{n\text{MAX}}$, \bar{I}_n and σ_n are respectively the minimum and maximum values, the average and the standard deviation of the n -th input feature of the dataset.

Both techniques help neural networks learn faster by ensuring that gradients propagate more uniformly through the layers, reducing the likelihood of vanishing or exploding gradients [38]. Additionally, normalization can prevent the network from favoring certain

features simply because they have larger numerical values, thereby improving the overall generalization of the model.

In this work, both techniques were tested. However, min-max normalization consistently produced better results and has therefore been chosen as the normalization method for this study.

2.1.2.1.3 Transformation

Certain input features of a neural network may exhibit highly non-linear behavior, where a small subset of relevant samples have values that deviate substantially from the average, leading to an uneven distribution across the dataset. This uneven distribution of input feature value within the dataset can hinder the training process and compromise its accuracy. To address this imbalance and facilitate the training process, various transformation functions can be applied to the input features prior to normalization. One common approach is power scaling, defined as:

$$s(I_n) = \text{sgn}(I_n) (|I_n|)^p, \quad (2.2)$$

where p exponent can be selected based on what is deemed most suitable for the specific input feature I_n . However, this scaling function has the drawback of zero or infinite slope for zero valued input feature (if exponent $p \neq 1$). This leads to a grouped or spaced distribution of samples in the near-zero region, which could result in poor performance. More advanced approaches have also been explored, including the Yeo-Johnson power transformation [129]

$$s(I_n, \lambda) = \begin{cases} \frac{(I_n+1)^\lambda - 1}{\lambda} & \text{for } \lambda \neq 0, I_n \geq 0 \\ \log(I_n + 1) & \text{for } \lambda = 0, I_n \geq 0 \\ -\frac{(1-I_n)^{2-\lambda} - 1}{2-\lambda} & \text{for } \lambda \neq 2, I_n < 0 \\ -\log(1 - I_n) & \text{for } \lambda = 2, I_n < 0 \end{cases}. \quad (2.3)$$

This transformation scales the dataset in order to obtain a sample distribution close to a normal distribution, with the transformation hyperparameter λ determined by an optimization problem. However, this scaling can excessively clip the transformed input features for the rarest samples, potentially leading to poor model performance when such conditions occur in real-time applications.

The last tested scaling function is

$$s(I_n) = \text{sgn}(I_n) \log(p|I_n| + 1), \quad (2.4)$$

where the coefficient p can be chosen according to what is most appropriate for the particular dataset. It expands the range of values for near-zero samples while maintaining a satisfactory distribution for the rarest samples.

An example of the application of the different scaling functions to the input feature I_n is presented in figure 2.2. This figure shows the input feature and its scaled version plotted. To ease comparison, all plotted values are normalized through min-max scaling.

2.1.2.2 Neural network training

This section outlines the general neural network architecture, introduces the selection of the loss function, presents a weighting strategy to tackle the challenges of imbalanced

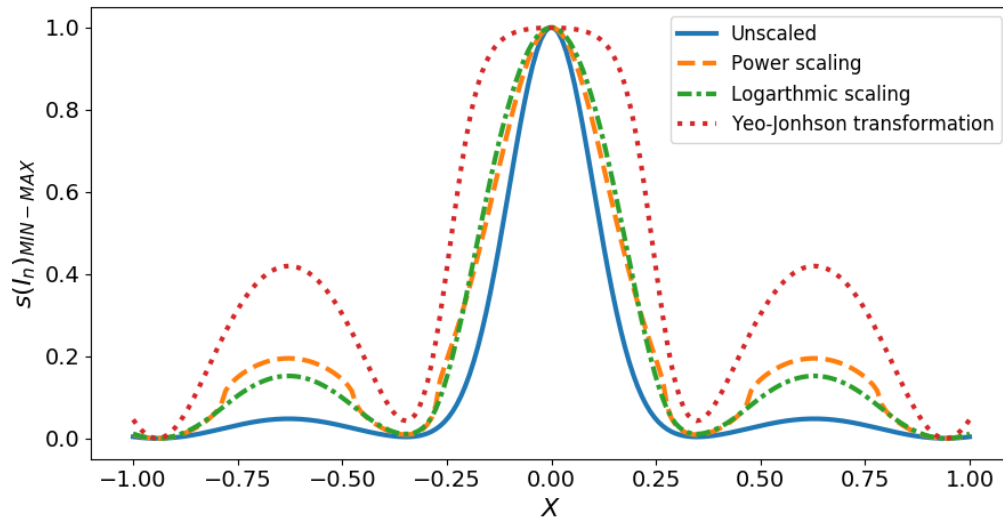


Figure 2.2: Example of input feature transformation. Comparison between scaled and unscaled input feature I_n .

datasets, and discusses the optimizer choice and training methodology. All these elements are then employed along the following of the work.

2.1.2.2.1 Neural network architecture

All the neural networks employed in the work process a set of input features and generate a single output. The neural network comprises an input layer, where the input features are fed; multiple hidden layers; and an output layer, which yields the estimated quantity. The number of nodes in the input and output layers is determined by the dimensionality of the input feature set and the number of quantities to be estimated, respectively.

Moreover, to account for the varying value ranges across the input features in the datasets, a normalization layer is added between the input and hidden layers. This layer integrates the min-max normalization, as described in equation (2.1) and used during training, into the neural network inference, simplifying the integration of the neural network into the CFD solver.

These normalization nodes are connected to their respective input features and are fully connected to the subsequent hidden layer. Prior to neural network training, the weights and biases of the normalization layer are determined to normalize each input feature (i.e., scaling to a range from 0 to 1) based on the training dataset.

A schematic representation of the general architecture of the employed NN employed in the work is given in figure2.3.

2.1.2.2.2 Loss function

The proposed wall models are data-driven, meaning the neural network is built upon training by data extracted from wall-resolved RANS simulations. The NN training optimizes weights $w_{i,j}^{(L)}$ and bias $b_i^{(L)}$ for all neurons by minimizing a loss function ϵ that evaluates the error between the CFD dataset and NN predictions. The choice of the loss function is thus fundamental, as it directly influences how the model learns and optimizes its parameters. A well-suited loss function can accelerate convergence and improve accuracy, while an inappropriate one may lead to poor performance or slow

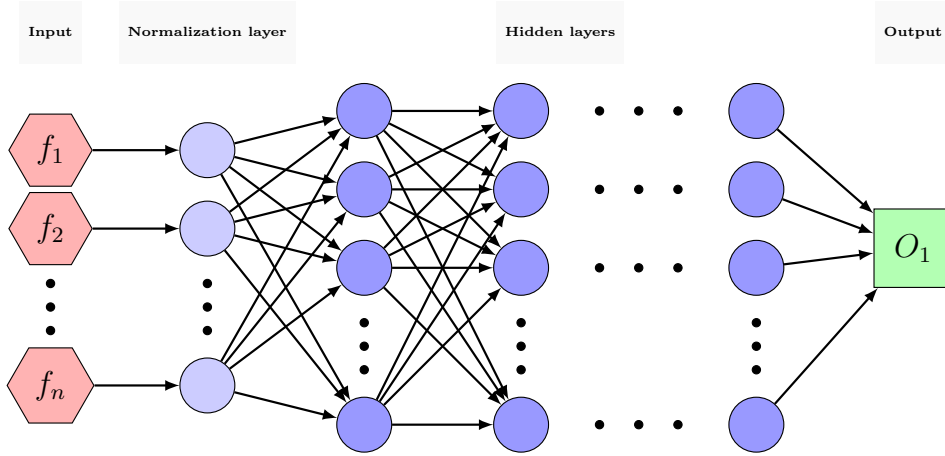


Figure 2.3: Schematic representation of the general architecture of the employed NN

training. Consequently, during the work, multiple loss functions are tested to determine the best option, ensuring the selected one yields the most accurate results compared to the desired outcomes. Throughout the work, all tested loss function options are presented, showcasing the process of selecting the one that provides the best results in comparison to the desired outcomes.

2.1.2.2.3 Sample weighting

To optimize training, data imbalance in the regression dataset is considered. Data imbalance occurs when certain samples values dominate the dataset, leading to biased model performance. This can result in the NN focusing on predictions near the dominant range, reducing its ability to generalize to underrepresented ranges. Consequently, the model may struggle to accurately predict values from these less frequent ranges due to insufficient training examples.

To address imbalance issues, techniques like stratified sampling [121] or weighted loss functions [125] can be applied. In this case, the latter approach is chosen. The loss function is thus defined as

$$\epsilon = \frac{1}{N_S} w_{\rho_i} \sum_{i=1}^{N_S} f(\mathbf{O}_i, \hat{\mathbf{O}}_i), \quad (2.5)$$

in which, the coefficient w_{ρ_i} is a weighting scalar that accounts for the sample distribution in the training dataset. This term corrects learning issues that appear when the training samples are unevenly distributed: without the scaling w_{ρ} , subsets of the dataset where the sample distribution is dense are artificially favored since the network attempts to minimize an average error over all samples. A lower sample coefficient is computed for common samples across the training dataset, while rarer ones get higher values of the weighting coefficient.

Throughout this work, two main approaches were tested, respectively, in chapters 3 and 4. The first approach follows a method similar to that of Zhou, He, and Yang [131], where the weighting coefficient w_{ρ} is calculated as inversely proportional to the local sample density (i.e. the probability density function of the sample distribution) ρ_i within the dataset:

$$w_{\rho i} = \frac{1}{\rho_i}. \quad (2.6)$$

The second approach, proposed by Steininger et al. [107] sees the weighting coefficient computed as

$$w_{\rho i} = \frac{\max(1 - \alpha_\rho \hat{\rho}_i, \epsilon_\rho)}{\frac{1}{N^S} \sum_{k=1}^{N^S} \max(1 - \alpha_\rho \hat{\rho}_k, \epsilon_\rho)}, \quad (2.7)$$

where $\hat{\rho}$ denotes the min-max normalized local density of the training dataset. The hyperparameters α_ρ and ϵ_ρ allow to tune the weighting scheme. They are selected respectively equal to 1 and 10^{-16} for this work. The parameter α_ρ allows for the tuning of the importance of the weighting scheme. When $\alpha_\rho = 0$, uniform sample weights are obtained, whereas higher values of α_ρ accentuate the effects of the weighting scheme. Meanwhile, the hyperparameter ϵ_ρ sets a lower bound for the weighting coefficient, ensuring that no data points are weighted negatively, since models would try to maximize the difference between estimate and reference value for these data points during the training process. Additionally, it is imperative to prevent any weight from reaching zero in order to avoid models ignoring parts of the dataset. Moreover, as it is explicitly designed for gradient-based optimization, the weighting scheme ensures that the mean of all weights across the dataset remains unity, thereby preventing any undue influence of the learning rates of gradient descent optimizers throughout the training process.

Both proposed approaches rely on the local sample density ρ_i of the dataset, which is computed using a kernel density estimation (KDE) technique [99]. This involves estimating the dataset distribution by placing a kernel at each data point and summing these kernels to generate a smooth estimate. Consequently, the local density at the i -th sample is estimated as the superposition of kernels, each of them centered on the k -th sample:

$$\rho(\mathbf{x}_i) = \frac{1}{N^S} \sum_{k=1}^{N^S} K_\sigma(\mathbf{x}_i - \mathbf{x}_k). \quad (2.8)$$

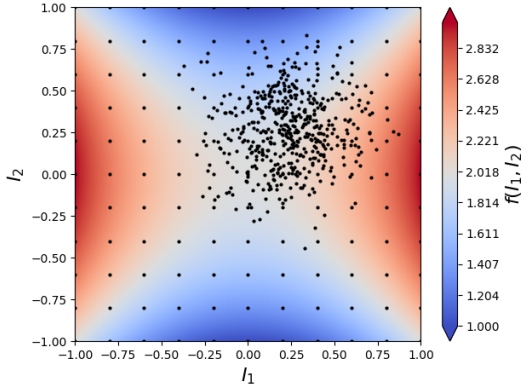
Here, the K_σ represents the chosen kernel functions. While \mathbf{x} denotes any sample from the dataset, consisting of vectors with a length of $n + 1$, considering n input features I along with the output value. The density calculation is performed using Gaussian distributions with unitary standard deviation as kernel on standardized training dataset (i.e. zero mean and unitary standard deviation).

An example of imbalanced training is given in figure 2.4, in which a neural network is trained to reproduce the functional $f_{NN}(I_1, I_2) = I_1^2 - I_2^2$. The post-training results illustrate significantly better accuracy in regions where data samples are concentrated when the weighted loss approach is not applied, whereas the accuracy is more uniform across the dataset when sample weighting is used.

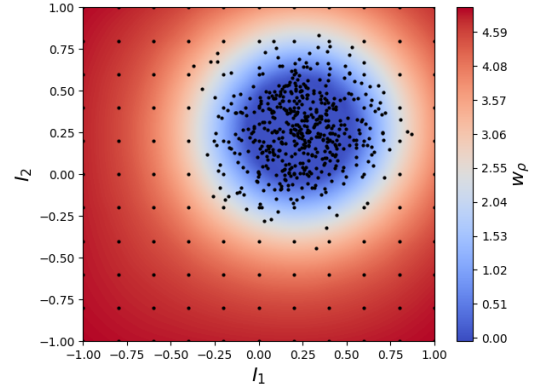
2.1.2.2.4 Optimizer and training strategy

This section outlines the training process for the neural network employed in this work. The training is conducted using the Adam algorithm, introduced in section 1.5.2.3, to minimize the neural network loss.

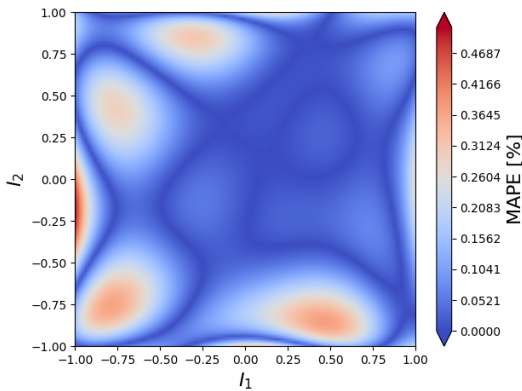
In order to evaluate the possibility of overfitting, the dataset is split in training dataset and validation dataset. The training dataset comprises 85% of the available data, while the validation dataset consists of the remaining 15%. During each epoch, the NN and the



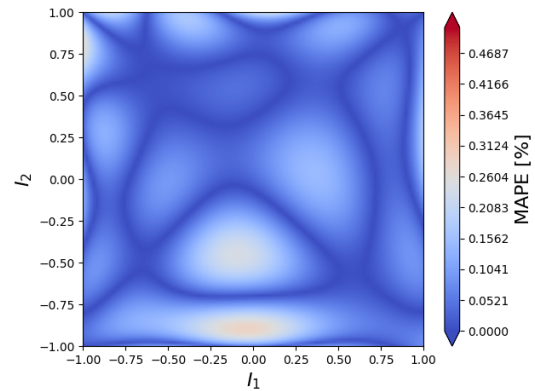
(a) Target function and samples of the training dataset.



(b) Sample weighting coefficient w_ρ and samples of the training dataset.



(c) Mean Absolute Percentage Error (MAPE) of the learned function without sample weighting.



(d) Mean Absolute Percentage Error (MAPE) of the learned function with sample weighting.

Figure 2.4: Representation of the sample weighting during neural network (NN) training. NN trained to reproduce the functional $f_{NN}(I_1, I_2) = I_1^2 - I_2^2$ on an imbalance dataset. Comparison between the weighted loss function and the unweighted one.

descent algorithm are supplied with the entire training dataset, divided into mini-batches of 16 samples. Every epoch, the samples contained in any of the mini-batches are shuffled.

The weights and biases of the neural networks are initialized using the He normal initialization [47].

The initial learning rate for the Adam algorithm is set at 0.001 and, during the neural network training, the convergence of the optimization algorithm benefits of a gradual reduction in the learning rate as epochs progress. The learning rate reduction is guided by the loss value. Specifically, the learning rate is reduced by 20% until it reaches a minimum of 10^{-8} . Each reduction occurs if the minimum loss value achieved over the previous 40 epochs remains unchanged.

A validation stopping criterion is used to stop training if the validation loss does not decrease by 10^{-4} over the last 400 iterations. At the end of the training process, the weights and biases that resulted in the lowest validation loss are selected for the final neural networks model.

2.1.3 CFD solver

The choice of the CFD solver was primarily driven by the need for easy access to the underlying source code to effectively integrate neural network utilization. Consequently, the experimental solver FAST Structured (FastS) [72], which has been under development at ONERA since 2015, was favored over other in-house solvers. FastS is a finite-volume solver dedicated to the numerical simulation of compressible turbulent flows. It is developed in Fortran and C++ for low- and mid-level functions, respectively, and wrapped in Python to facilitate user interactions. FastS is a cell centered solver based on a second-order finite volume approach for block-structured meshes. A key feature of this solver is its ability to efficiently handle various steady and unsteady simulations, enabling the update of 10 million cells per second on a single Intel Broadwell processor. These impressive performance metrics are achieved through a hybrid MPI/OpenMP parallelization, combined with various innovative HPC approaches, such as memory access optimization techniques (cache blocking) and vectorization [2]. FastS also features a dedicated Cartesian solver. Additionally, it integrates closely with the Cassiopée Python tool [8], enabling seamless pre- and post-processing via the CGNS standard [114, 94].

2.1.3.1 Numerical approach

This section outlines the numerical approach employed to solve the RANS equations for wall-resolved simulations used to generate the training datasets, as well as the simulations for validation and testing purposes, which are necessary to assess the performance of the wall model.

For both data collection and testing, the convective fluxes are discretized using the Roe flux scheme [91], extended to third order through a MUSCL strategy, as showed in section 1.2. The steady-state solution of the RANS equations is computed using an implicit temporal integration scheme, with a local time-stepping method and a CFL number set to 20.

All flow computations in this work, unless stated otherwise, are initialized with a constant field corresponding to free-stream conditions and are considered converged when the residual norms decrease by at least six orders of magnitude.

2.1.3.2 Neural network integration

The CFD solver has been enhanced to support online NNs inference during the solution process. Initially, an interface between the Python ML framework and the Fortran routine for the boundary layer application was considered. However, this approach is generally less computationally efficient than a direct Fortran implementation [29], which was ultimately adopted.

The capability for NN inference is now integrated within the Fortran source code of the CFD solver, FASTS. As shown by equations (1.70) and (1.71), Fully Connected Neural Network are a chain of matrix operations followed by a nonlinear vector function performed between the inputs and outputs of each NN layer. Therefore, generating a code that reproduces the learned functional is straightforward for given set of network parameter. Consequently, in the code a routine capable of reproducing a generic NN has been implemented. Moreover, the CFD solver has been modified to read the NN parameters (architecture, weights, biases, and activation functions), fed during the configuration of the simulation case, and configure the Fortran routine to replicate the required model. To improve computational performance in NN inference, matrix operations were intended to

be implemented using the Intel Math Kernel Library, which is expected to offer a more efficient computation than basic Fortran matrix operations. However, this approach turned out to be less efficient, particularly for relatively small models. As a result, built-in Fortran operations are utilized instead. The NN data is stored within the solver's data structures, and the weight and bias matrices are assembled prior to apply the wall model.

As discussed in chapter 3, the derivatives of the neural network with respect to its inputs are required. These derivatives are computed using the back-propagation algorithm, described in section 1.5.2.1, starting the backward pass from the outputs nodes of the NN. The development of the differentiation routine in Fortran was facilitated by Tapenade [53], a source-to-source automatic differentiation tool.

2.2 Wall model strategy

This section aims to introduce the overall strategy of the wall model and explain its implementation within a finite volume framework. In the following chapters, this general strategy is tailored to the specific formulation of the model, with additional details provided on its formulation and implementation.

Given a wall surface Γ_W that forms the boundary of the domain Ω , let us introduce an interface Γ_I within the domain. As shown in figure 2.5, the present wall law aims at reconstructing the flow state in the portion of the domain Ω_M and transfer the boundary condition from the surface Γ_W at a point P to an equivalent boundary condition on the surface Γ_I at a point I , moving along the wall normal direction from P .

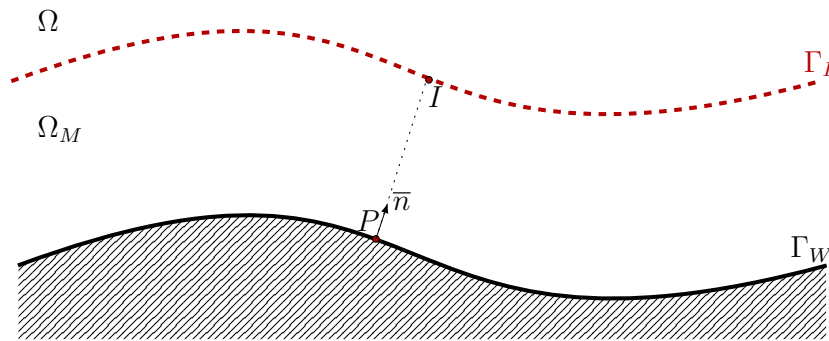


Figure 2.5: Scheme of wall model strategy

2.2.1 Wall model strategy in a finite volume framework

In a numerical framework, the model involves replacing the computation of the near-wall solution with a modeled evolution in Ω_W which shifts the boundary condition at the wall to the boundary Γ_I of the modeled region. The simulation domain is split into the near-wall region Ω_W , where the wall model is employed, and the rest, where conventional RANS integration occurs. These regions are separated by the interface Γ_I , called as RANS - Wall model interface (R-WM I.) in the following, which acts as a shifted domain boundary with special boundary conditions defined by the wall model. A common choice is to define the region Ω_W as a defined number of cell layers in the near wall region. This sets the R-WM I.

along the upper edge of the furthest cell layers belonging to the modeled region, as shown in figure 2.6. Consequently, in the considered cell-centered framework, a ghost cell method is applied to set the boundary condition at point I , located at the edge of the first layer of RANS-integrated cells. This method is introduced in the following section. Additionally, the application of the wall model requires knowledge of the local flow state at a point along the wall-normal direction from point P . This is accomplished by using a sampling point S within the RANS-integrated region above the interface. This point is positioned at a cell center along the wall-normal direction from point P , as illustrated in figure 2.6.

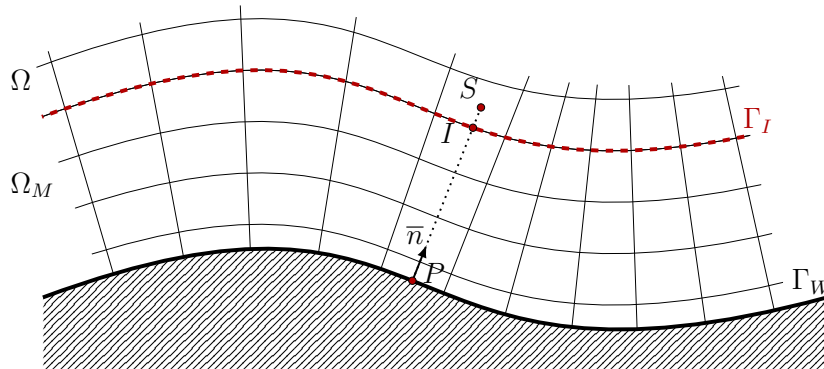


Figure 2.6: Scheme of wall model strategy in a discrete cell centered framework

2.2.2 Ghost cells approach

In a finite volume framework, the boundary conditions for RANS integration may be set by modeling the flow values at ghost cells, which are cells within the numerical stencil of the RANS region located across the interface (i.e. in the modeled Ω_W portion of the domain). The number of ghost cells required depends on the numerical scheme: in this study, two layers of ghost cells are required to set the boundary conditions across the interface. See section 2.1.3 for more details about the numerical scheme adopted.

This is illustrated in figures 2.7 and 2.8, where the modeled ghost cells are highlighted. A representative resolved cell, denoted by a black dot, and its numerical stencil represented by a dashed line are shown. Cells below the ghost cells can be ignored, since they do not affect the RANS computation: they lie beyond the numerical scheme's influence on the RANS region. During simulations, these cells are discarded to save computational time. However, post-convergence, they are nonetheless computed using the wall model to reconstruct the near-wall flow (for post-processing purpose).

Two situations exist for the R-WM I.. In the first configuration, illustrated in figure 2.8a, the interface aligns with the wall direction, and the model determines the value of the layers directly beneath the interface. The second configuration is depicted in figure 2.8b. This scenario occurs when the model is applied to a segment of the wall surface or when an edge of the wall surface is present. In this configuration, to fully populate the ghost cells, the model extends from the wall to the upper R-WM I. for the two adjacent cell columns.

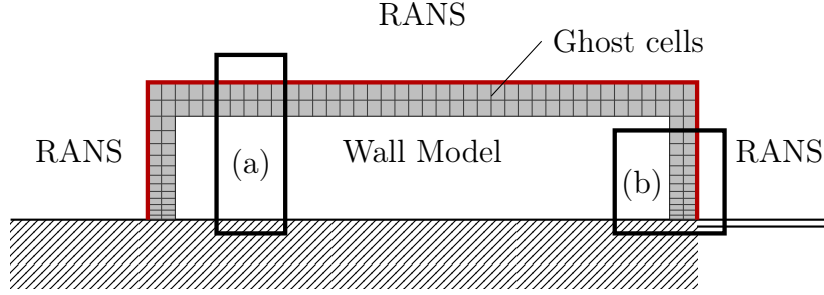


Figure 2.7: Schematic representation of modeled zone and RANS resolved ones. The RANS-Wall model interface is in red. The modeled cells (ghost cells) are shown in gray. Details given in figure 2.8.

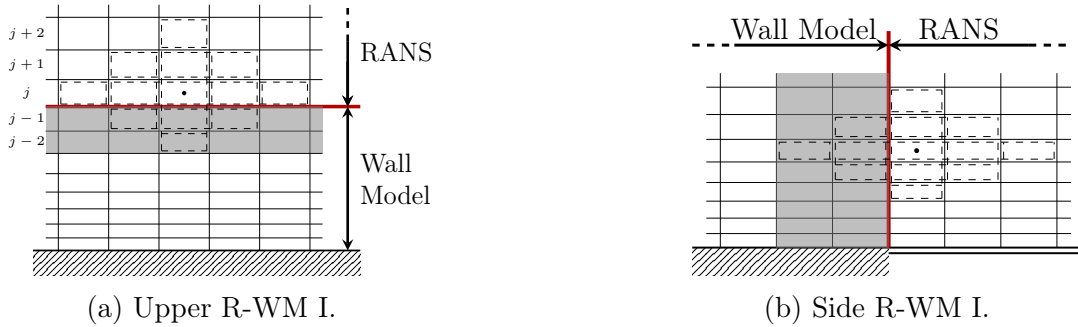


Figure 2.8: Grid structure used for RANS simulations coupled with wall modeling. The RANS - Wall model interface (R-WM I.) is in red. The modeled cells (ghost cells) are shown in gray. The numerical stencil of a representative RANS resolved cell nearby the modeled area is delimited by dashed lines.

2.2.3 Model components

The wall law models the flow state in the subdomain Ω_M . It comprises three distinct components: a functional model of the wall tangent velocity evolution, based on a data-driven approach; a physical model governing the thermodynamic state and the wall normal velocity field; a model for the turbulent viscosity in the near-wall region. These components are presented in the following chapters.

2.3 Main test cases

This section introduces the flow configurations used for training and testing the model. It begins by detailing the geometry of the cases, followed by a description of the domain setup and flow characteristics. Finally, the numerical discretization of the domain for each case is outlined.

The main flows discussed here include a two-dimensional bump and an airfoil case, which are described in the following sections.

2.3.1 2D Bump case

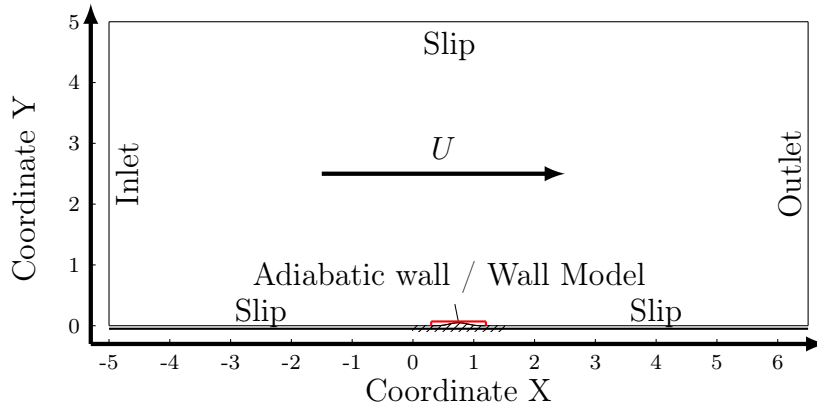
The main flow configuration used for the data-based wall model training and evaluation is inspired by a documented test case from NASA [95]. In the following, (X, Y) designates

the Cartesian coordinate system associated with computational domain configuration, while (x, y) refers to the local wall-tangent and wall-normal system.

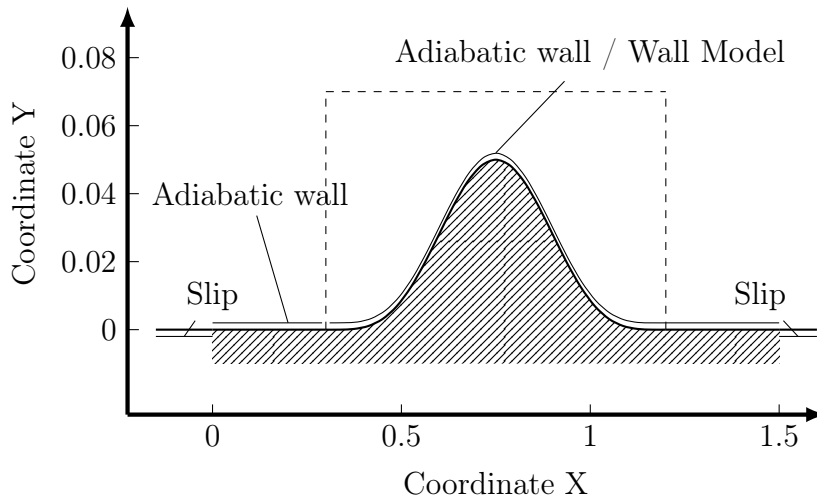
The geometry is defined as

$$Y(X) = \begin{cases} h \cdot \sin\left(\frac{\pi}{0.9}X - \frac{\pi}{3}\right)^4 & 0.3 \leq X \leq 1.2 \\ 0 & 0 < X < 0.3 \text{ and } 1.2 < X < 1.5 \end{cases}, \quad (2.9)$$

where h is the height of the bump. Equation (2.9) is applied for $X \in [0, 1.5]$, which corresponds to the wall extent. The resulting shape is shown in figure 2.9b. The simulation domain spans $X \in [-5, 6.5]$ and $Y \in [0, 5]$, as shown in figure 2.9a, where the chosen boundary conditions are also depicted. For the reference simulation, used to build the training dataset, the adiabatic wall condition is applied along all the wall surface, while, for testing configurations, the wall model boundary condition replace the adiabatic wall condition for $X \in [0.3, 1.5]$, as shown in 2.9b. Slip boundary conditions are applied upstream and downstream from the adiabatic wall, and on the upper boundary of the simulation domain. An inlet and an outlet boundary condition is respectively chosen for the upstream and downstream boundaries.



(a) Domain and boundary conditions for the 2D bump case.



(b) Details of the 2D bump case setup and its boundary conditions at walls.

Figure 2.9: Bump case: Simulation domain, boundary conditions and geometry details.

Several flow configurations are utilized for training and testing the wall model. These configurations are primarily generated by adjusting the Reynolds number Re of the flow and bump height h , the latter affecting the streamwise pressure gradient.

Each flow configuration has the same free-stream Mach number ($M = 0.2$) and free-stream static temperature ($T_\infty = 300K$). The Reynolds number Re is calculated using free-stream quantities and a reference length $L = 1 m$. Since the free-stream Mach number and temperature are fixed, changes in the flow Reynolds number are achieved by adjusting the free-stream density. All quantities presented in the following are non-dimensionalized using the reference length L , the free-stream velocity U_∞ , temperature T_∞ , and density ρ_∞ .

2.3.1.1 Domain discretization

A fine-structured grid is used to obtain reference data for both training and testing cases. From $X = 0.3$ to $X = 1.2$, the grid is uniform along the curvilinear coordinate x . Over the bump, 100 computational points are placed, giving a $\Delta x \approx 0.01$. Upstream of the bump, in the zone ranging between $X = 0$ and $X = 0.3$, the mesh is refined following an exponential distribution in the streamwise direction: the smallest cells located at the leading edge start with a spacing $\Delta x \approx 0.001$, which progressively increases to conform to the bump region at $X = 0.3$. The downstream part of the mesh (i.e., in the zone from $X = 1.2$ and $X = 1.5$) mirrors the upstream one such that the full grid is symmetrical with respect to the bump geometry. Finally, the streamwise structure of the mesh is completed from the bump geometry to the edges of the simulation domain, with a growing mesh spacing characterized by a growth ratio of 10%. In the wall-normal direction, the grid develops normally to the wall surfaces. The first computational point is placed at the same distance from the wall for all streamwise locations, giving a dimensionless wall distance between $y^+ = 0.01$ and $y^+ = 0.6$ along the bump for all the studied flow configurations. A 5% growing ratio for the cell size is used in the wall-normal direction.

Figure 2.10 presents various perspectives of the meshes of the 2D bump case, including a detailed close-up near the bump surface.

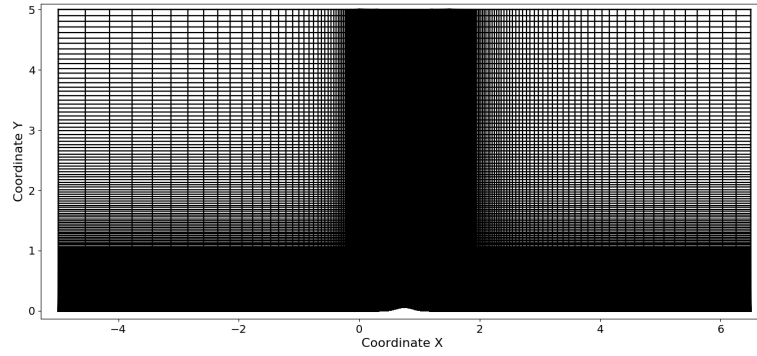
2.3.2 Airfoil case

In order to assess the robustness of the proposed wall models, we will also use in this work a completely different geometry from the bump used during the training procedure. Specifically, the model will be tested on the Wortmann airfoil FX60-100.

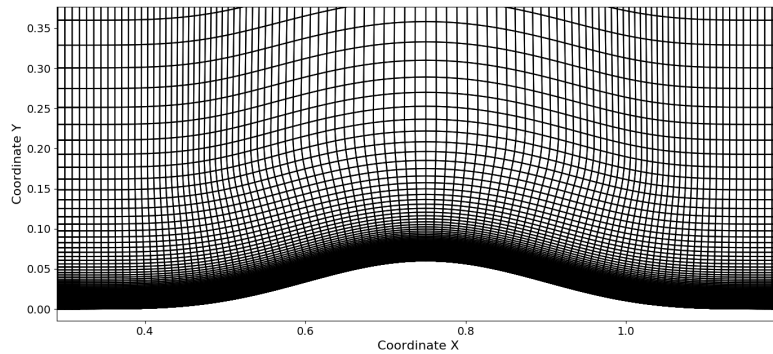
The computational domain, depicted in figure 2.11a, encompasses the airfoil with a unitary chord length ($c = 1 m$) serving as the reference. The airfoil leading edge is placed at $(X, Y) = (0, 0)$, with the simulation domain composed of a semicircle of radius equal to 10 times the chord centered at the leading edge. This is followed, in the streamline direction, by a rectangular shape spanning to $X = 15$.

The outer boundaries are defined by a characteristics-based far field boundary condition. Along the airfoil surface, an adiabatic wall condition is applied for reference cases. For testing cases, adiabatic wall condition is applied at the leading edge, then the wall model is applied downstream, as illustrated in figure 2.11b. More information about wall boundary conditions and model application are given in the results section 4.5.2.

Each flow configuration is characterized by a free-stream Mach number of $M = 0.2$ and temperature of $T_\infty = 300K$. As for the bump case, the different Reynolds numbers Re , which are computed through free-stream quantities and a reference chord $c = 1 m$, are obtained by modulating the free-stream density ρ_∞ . Reynolds numbers of $Re = 6 \cdot 10^6$ and 10^7 are set for the presented cases, with an angle of attack fixed at $\alpha = 0^\circ$.



(a) Domain and boundary conditions for the 2D bump case



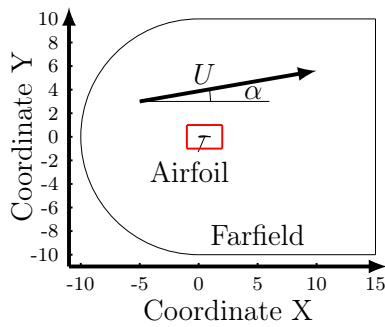
(b) Details of the near wall domain discretization for the 2D bump case

Figure 2.10: 2D Bump case: Domain discretization.

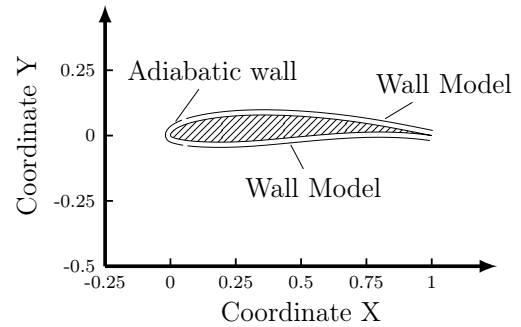
2.3.2.1 Domain discretization

The computational domain is discretized by a structured grid. The airfoil surface is defined by a bi-geometric distribution of computational points. The smallest cells are located at the leading and trailing edges of the airfoil and present a spacing of $\Delta x \approx 0.001$ which progressively increases with a growth ratio of 5%. A total of 163 computational points are considered on the airfoil. In the wall-normal direction, the initial computational points maintain a consistent distance from the wall across all streamwise positions, resulting in a dimensionless wall distance below $y^+ = 0.8$ along the airfoil for both cases. A growth ratio of 2% is applied to the cell size in the wall-normal direction.

An overview of the domain discretization of the Wortmann airfoil case is given in figure 2.12, including a close-up of the near wall mesh around the airfoil.

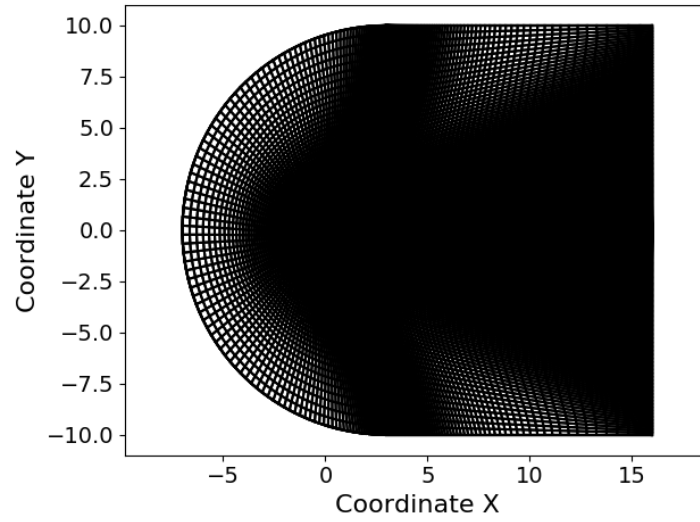


(a) Domain and boundary conditions for turbulent flow over the Wortmann FX60-100 airfoil.

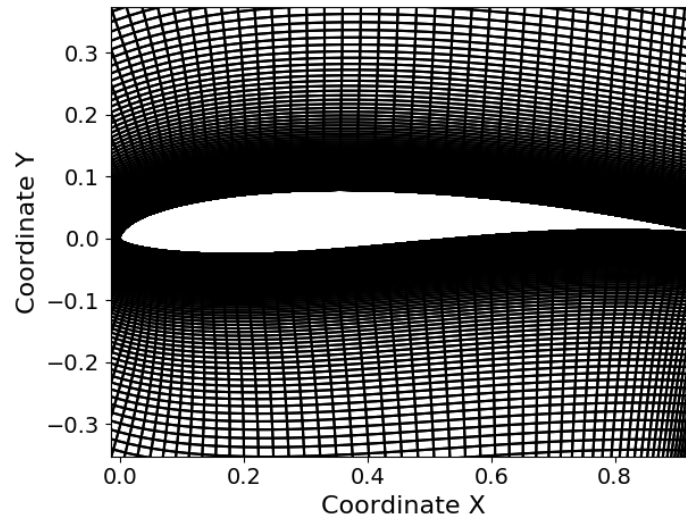


(b) Details of the Wortmann FX60-100 airfoil case setup and its boundary conditions at walls.

Figure 2.11: Wortmann FX60-100 airfoil case: Simulation domain, boundary conditions and geometry details.



(a) Domain discretization for the Wortmann FX60-100 airfoil case.



(b) Details of the near wall domain discretization for the Wortmann FX60-100 airfoil case.

Figure 2.12: Wortmann FX60-100 airfoil case: Domain discretization.

Chapter 3

Data-driven wall models for RANS

3.1	Wall law formulation	64
3.1.1	Formulation for the wall tangent velocity evolution	65
3.1.2	Physical model for thermodynamic state and wall normal velocity field	65
3.1.2.1	Wall normal velocity	66
3.1.3	Near-wall Spalart-Allmaras modeling	66
3.2	Numerical implementation of the wall model	66
3.2.1	Iterative estimation of local wall shear stress	67
3.2.2	Wall model application	67
3.3	Flow configurations	68
3.4	Neural network implementation and training	71
3.4.1	Loss function definition	71
3.4.2	Neural network architecture and optimization	72
3.4.2.1	Optimization of the neural network architecture	73
3.4.2.2	Neural network architecture	74
3.4.3	Training and a priori results	75
3.5	Results	77
3.5.1	Test procedure	77
3.5.2	Global errors	78
3.5.3	Interpolation test results	78
3.5.4	Extrapolation cases	82
3.5.4.1	Flat plate case	83
3.5.4.2	Near separation case	83
3.5.4.3	Influence of dimensionless pressure gradient	83
3.5.5	Mass conservation	84
3.6	Conclusion	86

The existence of universal wall laws for turbulent boundary layers relies on boundary layer theory, scaling arguments, and dimensional analysis, introduced in section 1.3.2.2. When the flow quantities vary slowly in the streamwise direction, as compared to the wall-normal direction, the shape of the streamwise velocity profile $u_{\parallel}(y)$ becomes invariant in the inner region of the boundary layer when scaled with appropriate quantities, here the wall viscosity μ_w , the density ρ_w and the friction velocity u_{τ} , computed using the skin friction τ_w :

$$u_{\tau} = \sqrt{\frac{\tau_w}{\rho_w}} \quad \text{with} \quad \tau_w = \mu_w \left. \frac{\partial u_{\parallel}}{\partial y} \right|_{y=0}. \quad (3.1)$$

In the absence of streamwise pressure gradient, it is straightforward to show that:

$$u^+ = f(y^+), \quad u^+ = \frac{u_{\parallel}}{u_{\tau}} \quad y^+ = \frac{\rho_w u_{\tau}}{\mu_w} y, \quad (3.2)$$

where y^+ is the dimensionless wall distance and u^+ the dimensionless streamwise velocity. From dimensional analysis, it is then possible to show that $u^+ = y^+$ close to the wall and that u^+ follows a logarithmic profile for large y^+ in the inner region. In the presence of streamwise pressure gradients, these wall laws need to be adapted. A new non-dimensional parameter steps in, the streamwise wall pressure gradient:

$$u^+ = f(y^+, p^+), \quad p^+ = \frac{\mu_w}{\rho_w^2 u_{\tau}^3} \frac{\partial p}{\partial x}, \quad (3.3)$$

where x is the streamwise coordinate. This additional parameter has been introduced by Afzal [1], who gave an analytical expression for the evolution of the nondimensional streamwise velocity, as shown in equation (1.66). Even if Afzal's wall law includes the effects of the pressure gradient on the boundary layer evolution, equation (1.66) is valid only in the logarithmic region of the boundary layer. Additionally, it can only be applied for adverse (i.e., positive) pressure gradients due to the square root in the expression. In practice, this may cause implementation issues and possible convergence problems when trying to impose the wall model.

In the present chapter, contrary to Zhou, He, and Yang [131], we choose the dimensionless quantities (y^+, p^+) and u^+ for the input and output of our data-driven model, respectively. Neural networks are then used as universal interpolators on physically-scaled data to find a relation $u^+ = f(y^+, p^+)$. This straightforward approach is bound to provide more general results than a direct estimation of dimensional quantities. Given that there would exist a universal law $u^+ = f(y^+, p^+)$ for the near-wall region, once learned with enough data, this law could be used for any unseen geometry or unseen flow conditions.

The chapter describes in detail the practical implementation of such an approach and quantifies its performance in several test cases. It is organized as follows. First, the wall modeling approach is detailed in section 3.1, along with the numerical implementation in the CFD solver. Then, the flow configurations considered are presented in section 3.3. After, the implementation of the neural network employed is discussed in section 3.4, considering also an optimization strategy to define its architecture. The last section presents the results obtained with the data-based model on training and unseen configurations (interpolation and extrapolation conditions) before concluding.

This chapter is derived from the article published in Romanelli et al. [92].

3.1 Wall law formulation

As firstly introduced in section 2.2, the proposed wall model aims to reconstruct the flow state in the portion of the domain Ω_M , close to the wall, in order to transfer the no-slip boundary condition at a point P on the wall surface to an equivalent boundary condition at a point I far from wall and located on the wall normal direction from P . See figure 2.5.

This section introduce the main relations that are used to reconstruct the flow state in the portion of the domain Ω_M .

3.1.1 Formulation for the wall tangent velocity evolution

In the presence of streamwise pressure gradients, it is shown that a functional relation between u^+ , p^+ , and y^+ exist, as in equation (3.3). In developed form, it reads:

$$\frac{u_{\parallel}(y)}{u_{\tau}} = f\left(\frac{\rho_w u_{\tau}}{\mu_w} y, \frac{\mu_w}{\rho_w^2 u_{\tau}^3} \frac{\partial p}{\partial x}\right). \quad (3.4)$$

Consequently, the tangential velocity $u_{\parallel}(y)$ evolution can be easily determined as

$$u_{\parallel}(y) = u_{\tau} \cdot f\left(\frac{\rho_w u_{\tau}}{\mu_w} y, \frac{\mu_w}{\rho_w^2 u_{\tau}^3} \frac{\partial p}{\partial x}\right), \quad (3.5)$$

The quantities ρ_w , μ_w , and $\partial p/\partial x$ are estimated through a physical model presented in the following section. The skin friction velocity u_{τ} , directly linked to the wall shear stress, is obtained by solving the non-linear equation

$$g(u_{\tau}) = 0, \quad (3.6)$$

where

$$g(u_{\tau}) = f\left(\frac{\rho_w u_{\tau}}{\mu_w} y_I, \frac{\mu_w}{\rho_w^2 u_{\tau}^3} \frac{\partial p}{\partial x}\right) - \frac{u_{\parallel}(y_I)}{u_{\tau}}, \quad (3.7)$$

is obtained from equation (3.4). In the equation, the point marked as I indicates the point on Γ_I along the wall normal direction from the wall point P in figure 2.5.

3.1.2 Physical model for thermodynamic state and wall normal velocity field

The wall-normal temperature profiles in the Ω_M portion of the domain and along the Γ_I boundary (see figure 2.5) are modeled using the Crocco-Busemann's relation [80] adapted for adiabatic wall conditions:

$$T(y) = T_w - AU(y)^2, \quad \text{with} \quad A = \frac{T_w - T_e}{U_e^2}, \quad (3.8)$$

which sets the relation between the velocity magnitude $U = \sqrt{u_{\parallel}(y)^2 + u_{\perp}(y)^2}$ and the flow temperature T in the near-wall region. In equation (3.8), T_w is the wall temperature, T_e and $U_e = \sqrt{u_{\parallel}(\delta)^2 + u_{\perp}(\delta)^2}$ are the flow temperature and velocity magnitude outside the boundary layer of thickness δ (u_{\parallel} and u_{\perp} being respectively the tangential and normal velocity components with respect to the wall). As both A and T_w are unknown, they are determined enforcing the continuity condition of temperature profiles across the interface Γ_I . This is done differentiating (3.8) and solving the linear equation system

$$\begin{cases} T(y_I) = T_w - AU^2(y_I) \\ \left. \frac{\partial T}{\partial y} \right|_{y=y_I} = -2AU(y_I) \left. \frac{\partial U}{\partial y} \right|_{y=y_I} \end{cases} \quad (3.9)$$

for the given unknowns. The Crocco-Busemann temperature profile can be extended to estimate the temperature across the region Γ_W .

In the boundary layer, the wall-normal pressure gradient $\partial p/\partial y$ is close to zero. The density profile can thus be obtained using the perfect gas law with the temperature profile obtained from Crocco-Busemann's law combined with the reconstructed pressure field.

Additionally, since the temperature T and density ρ evolution are known, the wall density ρ_w and wall molecular viscosity μ_w can be determined through the chosen viscosity model (in our case, Sutherland's law).

Finally, the pressure p being constant in the wall-normal direction, the pressure gradient $\partial p/\partial x$ in the modeled region may be obtained by projecting the pressure gradient evaluated at the Γ_I interface along the tangent direction x .

3.1.2.1 Wall normal velocity

The wall normal velocity u_\perp is assumed to behave linearly in the Ω_M region, so that:

$$u_\perp(y) = Cy, \quad \text{with} \quad C = \frac{u_\perp(y_I)}{y_I}. \quad (3.10)$$

This approximation for u_\perp is non-conservative, and more elaborated handling of this component may be considered in future works. However, it does not represent an issue for the boundary layers configurations that are involved in the present work. The error introduced in the wall-normal direction u_\perp has been found to be negligible, since the main velocity component is tangential to the wall surface (see section 3.5.5 for the study of mass conservation issues).

3.1.3 Near-wall Spalart-Allmaras modeling

Kalitzin et al. [56] studied the Spalart-Allmaras variable $\tilde{\nu}$ in the near wall region of a quasi-equilibrium boundary layer. They showed that the behavior of the dimensionless S.-A. variable was defined as:

$$\tilde{\nu}^+ = \frac{\rho\tilde{\nu}}{\mu} \quad (3.11)$$

can be modeled as

$$\tilde{\nu}^+ = \kappa y^+, \quad (3.12)$$

in the inner region of the boundary layer (viscous sublayer and logarithmic layer). Here, $\kappa = 0.41$ is the Von Kármán constant. The presented wall models set the S.-A. variable $\tilde{\nu}$ to respect equation (3.12) in the Ω_M region. The accuracy of this model and its impact on the results are discussed in section 3.5.3.

3.2 Numerical implementation of the wall model

As seen in section 2.2.1, the model replaces the near-wall solution with a modeled evolution in the near-wall region Ω_W and shifts the boundary condition to the interface Γ_I , separating the modeled region from the rest of the domain, where conventional RANS integration occurs. In a discrete environment, Ω_W is typically defined as a fixed number of cell layers near the wall, with the interface Γ_I at the top of this region. The state of the flow at the interface (and at the point I) is unknown in a cell-centered framework and interpolation between the closest cells states is required. Alternatively, a sampling point S can be used above the interface, in the RANS region to obtain the local flow state useful to drive the model. Consequently, the point I at the interface location is replaced by the sampling point S in the wall model formulation presented above, in section 3.1. The adaptation of the wall model strategy in a discrete cell-centered framework is illustrated in figure 2.6.

The boundary conditions for RANS integration along the RANS - Wall model interface (R-WM I.) are set by modeling the flow values at ghost cells, which are cells within Ω_W region, but belonging to the numerical stencil of the RANS region located across the interface. The number of ghost cells required depends on the numerical scheme: in this work, two layers of ghost cells are required. The ghost cell configuration is illustrated in figures 2.7 and 2.8.

3.2.1 Iterative estimation of local wall shear stress

The skin friction velocity u_τ is obtained by solving the non-linear equation (3.7) applied to a sampling point S . Specifically, the tangential velocity $u_\parallel(y^S)$ and corresponding wall distance y^S at S are needed.

The solution to this nonlinear equation is numerically obtained by an iterative Newton-Raphson method that reads

$$u_\tau^n = u_\tau^{n-1} - \frac{g(u_\tau^{n-1})}{g'(u_\tau^{n-1})}. \quad (3.13)$$

The derivative g' is obtained analytically:

$$g'(u_\tau) = \frac{\rho_w}{\mu_w} y^S \cdot \frac{\partial f}{\partial y^+} - \frac{3\mu_w}{\rho_w^2 u_\tau^4} \frac{\partial p}{\partial x} \cdot \frac{\partial f}{\partial p^+} + \frac{u_\parallel(y^S)}{u_\tau^2}. \quad (3.14)$$

A guess for the skin friction velocity u_τ is required to initialize the Newton-Raphson procedure. For this, the simpler Werner and Wengle's wall law [122] is used, which can be reformulated as follows:

$$u_\tau(u_\parallel(y^S)) = \begin{cases} \sqrt{\frac{\mu_w u_\parallel(y^S)}{\rho_w y^S}} & \text{if } u_\parallel \leq \frac{\mu_w}{4\rho_w y^S} A^{\frac{2}{1-B}} \\ \left[\frac{1+B}{A} \left(\frac{\mu_w}{2\rho_w y^S} \right)^B u_\parallel(y^S) + \frac{1-B}{2} A^{\frac{1+B}{1-B}} \left(\frac{\mu_w}{2\rho_w y^S} \right)^{1+B} \right]^{\frac{1}{1+B}} & \text{otherwise} \end{cases}, \quad (3.15)$$

with $A = 8.3$ and $B = \frac{1}{7}$.

The iterative process is applied until the residual value of g approaches zero within a given tolerance (set in the following to 10^{-9}). Once the skin friction u_τ is obtained, equation (3.4) can be used to determine the wall tangent velocity $u_\parallel(y)$ at any height y in the modeled region: The Newton-Raphson procedure to compute u_τ is summarized in algorithm 1.

3.2.2 Wall model application

The section explains the methodology adopted to apply the wall model and determine the evolution of tangential velocity u_\parallel in the near-wall region of the boundary layer.

The process begins by determining the thermodynamic state within the near-wall modeled region and the evolution of the pressure field along the streamwise direction, as outlined in section 3.1.2. Next, the model estimates the local wall shear stress by solving,

Algorithm 1: Computation of u_τ

Inputs : $u_\parallel^S, y^S, \rho_w, \mu_w, \partial p/\partial x$
Initialize u_τ with equation (3.15)
Compute $g \leftarrow f\left(\frac{\rho_w u_\tau}{\mu_w} y^S, \frac{\mu_w}{\rho_w^2 u_\tau^3} \frac{\partial p}{\partial x}\right) - \frac{u_\parallel^S}{u_\tau}$ (equation(3.6))
while $|g| > 10^{-9}$ **do**
 Compute $g' \leftarrow \frac{\rho_w}{\mu_w} y^S \cdot \frac{\partial f}{\partial y^+} - \frac{3\mu_w}{\rho_w^2 u_\tau^4} \frac{\partial p}{\partial x} \cdot \frac{\partial f}{\partial p^+} + \frac{u_\parallel^S}{u_\tau^2}$ (equation (3.14))
 Update $u_\tau \leftarrow u_\tau - g/g'$
 Compute $g \leftarrow f\left(\frac{\rho_w u_\tau}{\mu_w} y^S, \frac{\mu_w}{\rho_w^2 u_\tau^3} \frac{\partial p}{\partial x}\right) - \frac{u_\parallel^S}{u_\tau}$ (equation(3.6))
end

through the iterative algorithm in 1, the non-linear equation (3.7) applied at the sampling point S .

Once the local shear stress has been determined, the tangential velocity u_\parallel in the near-wall region can be easily estimated through the equation (3.5) by just knowing the wall distance y of the modeled location. The figure 3.1 resumes the general methodology to compute $u_\parallel(y)$ within the modeled region.

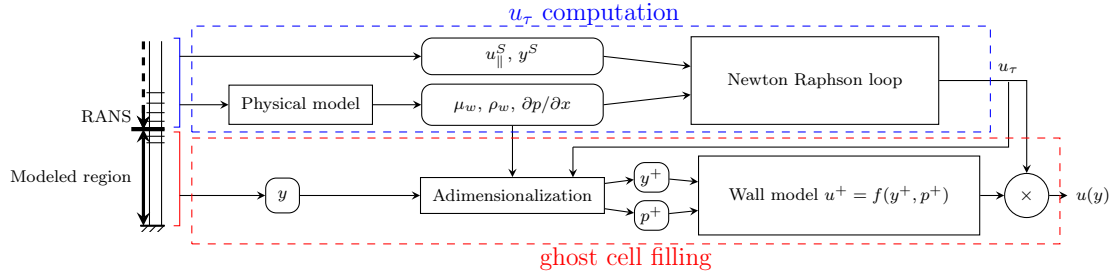


Figure 3.1: Schematic representation of wall model methodology to determine the tangential velocity profile within the modeled region through iterative approach.

3.3 Flow configurations

Reference and training data are obtained from a set of fine wall-resolved RANS simulations. The flow configuration adopted to train and test the current model is the 2D bump case, introduced in section 2.3.1.

Training and evaluation data are extracted for a well-established, fully turbulent boundary layer. Thus, the extraction zone for the datasets is restricted to the range $X = 0.3$ to $X = 1.2$ (corresponding to the bump region, see figure 2.9b. Various Reynolds numbers and bump heights h are considered to build the training and test datasets. The considered Reynolds numbers are $Re = 10^6$, $Re = 3 \cdot 10^6$, $Re = 6 \cdot 10^6$ and $Re = 10^7$, while the considered bump heights h are 0.05, 0.06, 0.07, and 0.08. They are selected to yield a weak to moderate pressure gradient along the geometry without inducing flow separation. A flow configuration with $h = 0$ (flat plate geometry) is also used as a test case.

The figure 3.2 shows the (Re, h) -combinations considered for the present chapter. The complete dataset, training and test sets, consists in 14 simulations. The training dataset (red dots) includes cases with moderate adverse pressure gradient p^+ (for instance, $h = 0.05$ and $Re = 6 \cdot 10^6$), but also a case with higher dimensionless pressure gradients approaching

flow separation ($h = 0.07$ and $Re = 10^6$). This may be seen in figures 3.3 and 3.4, which show respectively the skin friction coefficient C_f (almost reaching zero for one of the cases) and the non-dimensional pressure gradient p^+ (exhibiting a very strong value for vanishing C_f) of the training cases.

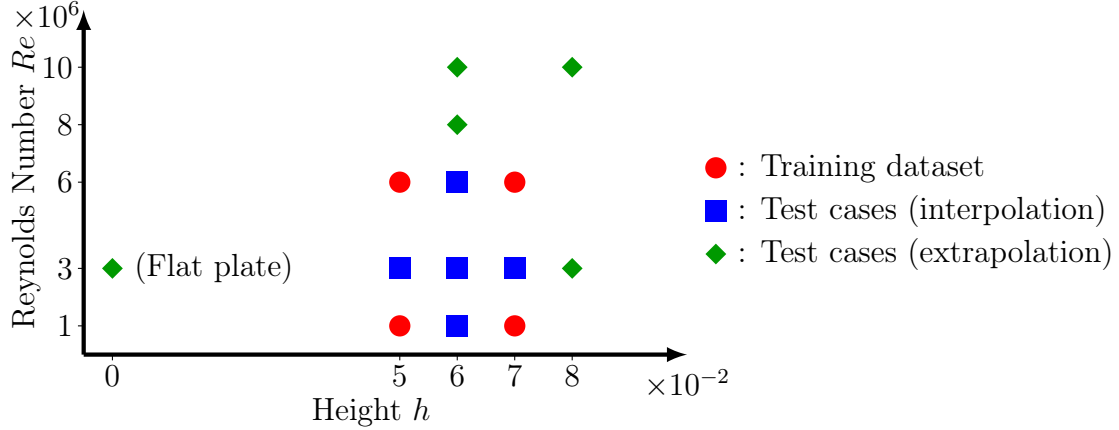


Figure 3.2: Training and test datasets obtained from different combinations of Reynolds number and bump height h .

The other configurations are used for testing. The test dataset contains five configurations that combine h and Re values within the range used for training (testing interpolation capabilities of the model in the (h, Re) -space), and five configurations that go beyond this range (testing extrapolation capabilities).

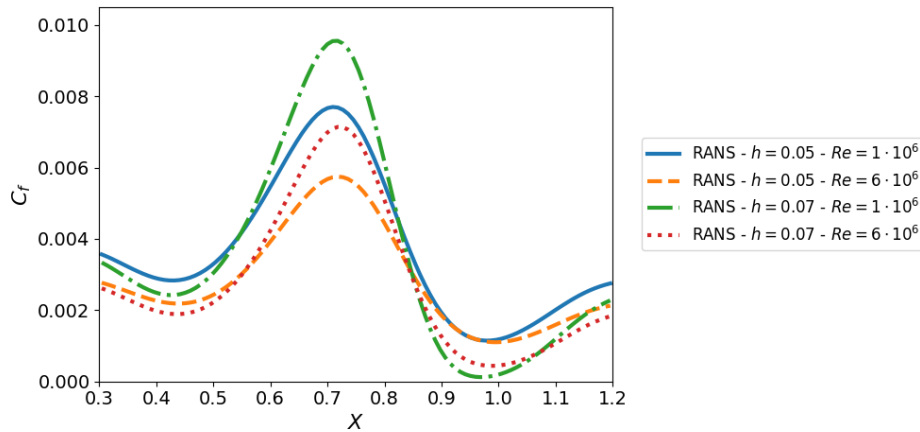


Figure 3.3: Streamwise evolution of the skin friction coefficient C_f for the four cases used for training.

The ratio between the bump height h and the boundary layer thickness δ at $X = 0.3$ for all the flow configurations is shown in table 3.1. Due to the presence of strong pressure gradients which affects the velocity profiles, the standard δ_{99} definition commonly used for flat plates cannot be applied here. Instead, the boundary layer edge and the associated boundary layer thickness δ are estimated through the vanishing of the shear stress and flow vorticity following the work of Cliquet, Houdeville, and Arnal [21]. The boundary layer edge can thus be estimated through an empirical estimation method, which relies on the boundary layer definition. At the boundary layer edge, the shear stress τ and the flow vorticity Ω must become small. The total shear stress can be defined as

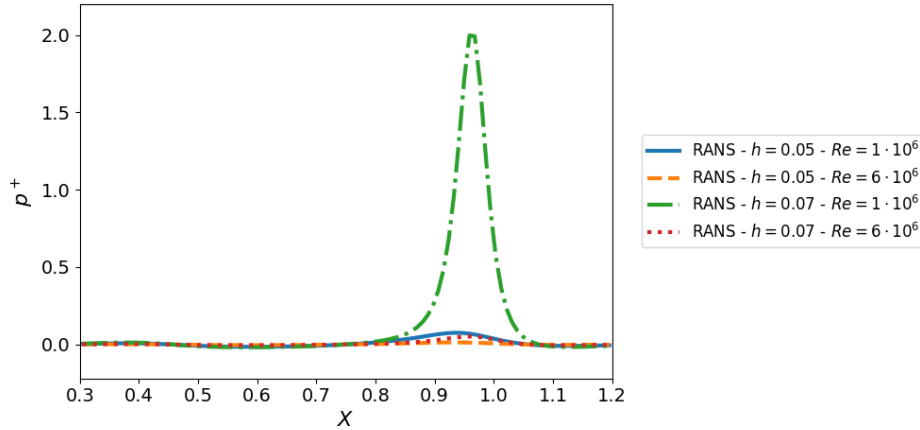


Figure 3.4: Streamwise evolution of the dimensionless pressure gradient p^+ for the four cases used for training.

$$\tau = \tau_l + \tau_v = \mu |\Omega| + \mu_t |\Omega|, \quad (3.16)$$

where the laminar component of shear stress τ_l and the turbulent shear stress τ_t are computed through the laminar and turbulent viscosity, respectively μ and μ_t .

The boundary layer thickness δ is defined such as

$$\delta = \min(\delta_\Omega, \delta_\tau), \quad (3.17)$$

where δ_Ω and δ_τ are the wall normal distances where ϵ_Ω and ϵ_τ reach respectively small empirical values as 0.001 and 0.015. The two parameters ϵ_Ω and ϵ_τ are defined as it follows

$$\epsilon_\Omega = \frac{|\Omega|}{|\Omega|_{max}} \quad \epsilon_\tau = \frac{|\tau|}{|\tau|_{max}}. \quad (3.18)$$

As an example, the estimated boundary layer thickness for the wall-resolved RANS case with $h = 0.06$ and $Re = 3 \cdot 10^6$ is represented over velocity magnitude contours in figure 3.5.

Table 3.2 shows the strongest non-dimensional pressure gradient for all flow configurations. It reveals that the case ($h = 0.08, Re = 3 \cdot 10^6$) involves p^+ values that exceed by two orders of magnitude the training values (thus explaining some convergence problems discussed later for this case).

	h/δ at $X = 0.3$				
	$h = 0$	$h = 0.05$	$h = 0.06$	$h = 0.07$	$h = 0.08$
$Re = 1 \cdot 10^7$			11.76		15.69
$Re = 8 \cdot 10^6$			11.54		
$Re = 6 \cdot 10^6$		9.26	11.11	12.96	
$Re = 3 \cdot 10^6$	0.0	8.62	10.16	11.86	13.56
$Re = 1 \cdot 10^6$		7.04	8.57	9.85	

Table 3.1: Ratio between bump height h and boundary layer thickness δ at $X = 0.3$ for all flow configurations

The inner region of the boundary layer, which is modeled by the present wall law, extends approximately up to $\frac{y}{\delta} \leq 0.15$. Consequently, training data, as well as the location

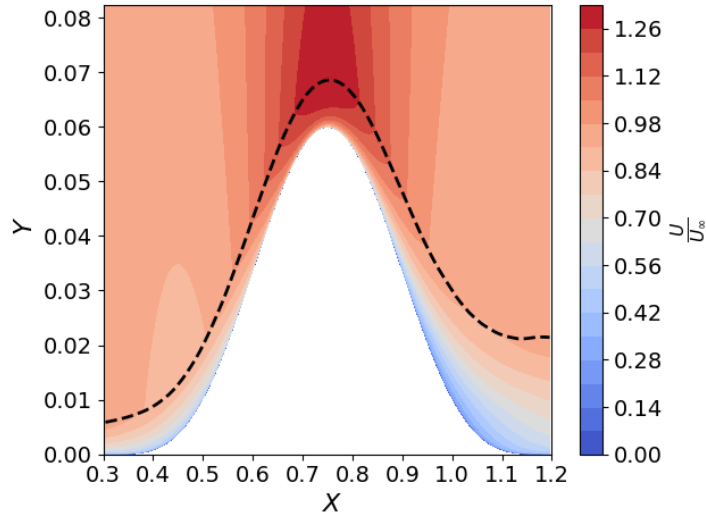


Figure 3.5: Velocity magnitude contours for the wall-resolved RANS case with $h = 0.06$ and $Re = 3 \cdot 10^6$. Boundary layer thickness represented with dashed line.

Strongest non-dimensional adverse pressure gradient p^+					
	$h = 0$	$h = 0.05$	$h = 0.06$	$h = 0.07$	$h = 0.08$
$Re = 1 \cdot 10^7$			0.014		0.099
$Re = 8 \cdot 10^6$			0.017		
$Re = 6 \cdot 10^6$		0.013	0.023	0.051	
$Re = 3 \cdot 10^6$	0.0	0.025	0.048	0.137	258.2
$Re = 1 \cdot 10^6$		0.076	0.187	1.997	

Table 3.2: Strongest non-dimensional adverse pressure gradient for all flow configurations

of the interface for the wall model during testing, need to be located within this inner region. The learning process is focused on samples satisfying $y^+ \leq 100$. This limit corresponds to the largest range of y^+ that is entirely contained within the 15% of the boundary layer thickness at each streamwise station.

3.4 Neural network implementation and training

A neural network is trained to estimate, from the dimensionless wall distance y^+ and the dimensionless pressure gradient p^+ , the dimensionless velocity u^+ as in equation (3.3). Moreover, the derivatives of the neural network, with respect to the inputs y^+ and p^+ , are needed for the Newton algorithm, as shown in equation (3.14).

This section outlines the steps involved in building the model, including the selection of the loss function, the choice of the model architecture, and the training results.

3.4.1 Loss function definition

The proposed wall model is data-driven, meaning the neural network is built upon training by data extracted from wall-resolved RANS simulations. The NN training optimizes $w_{i,j}^{(L)}$ and $b_i^{(L)}$ for all neurons by minimizing a loss function ϵ that evaluates the error between RANS data and NN predictions.

A first approach would be to consider the Mean Squared Error (MSE) for the loss function:

$$\epsilon_{\text{MSE}} = \frac{1}{N_S} \sum_{i=1}^{N_S} \left(f_{NN}(y_i^+, p_i^+) - u_{i,ref}^+ \right)^2, \quad (3.19)$$

where N_S is the number of samples. Yet, it puts more emphasis on the high u^+ values, since reducing the relative error of a given percentage becomes more advantageous for high u^+ values. Using the Mean Squared Relative Error (MSRE):

$$\epsilon_{\text{MSRE}} = \frac{1}{N_S} \sum_{i=1}^{N_S} \left(\frac{f_{NN}(y_i^+, p_i^+) - u_{i,ref}^+}{u_{i,ref}^+} \right)^2, \quad (3.20)$$

yields also difficulties for u^+ values close to zero, since it may diverge even for very small absolute errors. Additionally, the relation $u^+ = f(y^+, p^+)$ learned with the MSRE loss function was found to be wavy, which was problematic for the determination of the friction velocity within the Newton-Raphson algorithm.

The present approach followed the work of Park [84] to address the issues mentioned above, with a loss function based on a logarithmic expression of the error:

$$\epsilon = \frac{1}{N_S} \sum_{i=1}^{N_S} w_{\rho i} \left| \log \left(\frac{f_{NN}(y_i^+, p_i^+) + 1}{u_{i,ref}^+ + 1} \right) \right| + \frac{\lambda_2}{N_S} \|w^L\|_2. \quad (3.21)$$

The first term in (3.21) is the mean absolute logarithmic error between the NN output $f_{NN}(y_i^+, p_i^+)$ and the corresponding reference value $u_{i,ref}^+$ from the training dataset. This error function does not present the problems mentioned above. The coefficient $w_{\rho i}$ is a weighting scalar that accounts for the sample distribution in the training dataset, as introduced in section 2.1.2.2.3. As in the work of Zhou, He, and Yang [131], the samples weights are considered to be inversely proportional to the sample density, as in equation (2.6).

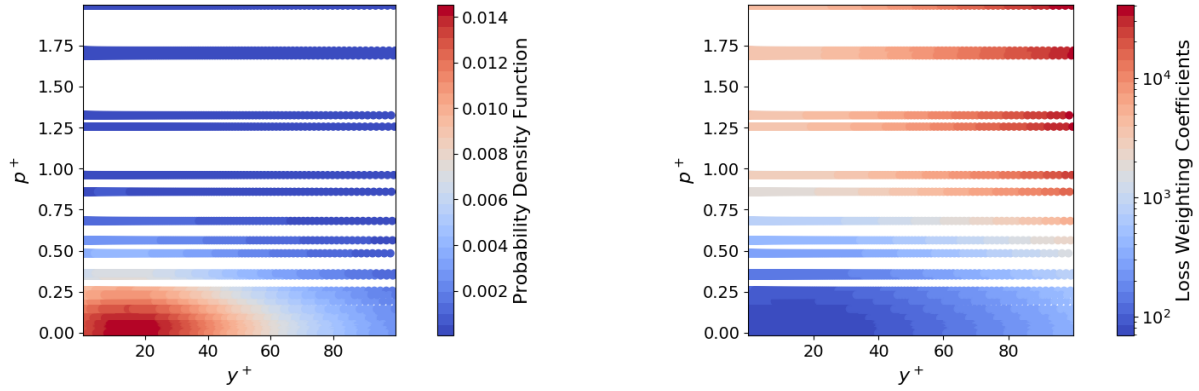
This term corrects learning issues that appear when the training samples are unevenly distributed: without the scaling w_{ρ} , regions in the (y^+, p^+) plane where the sample distribution is dense are artificially favored since the network attempts to minimize an average error over all samples. In our case, using w_{ρ} has shown improved results for high p^+ values, where there is a limited number of samples in the training dataset. Without this regularization, the network focuses on reducing the error for low p^+ values since it corresponds to most training samples.

Figure 3.6 shows both the sample density ρ_i of the dataset and the resulting coefficients $w_{\rho i}$.

The second term in the loss function ϵ is an L_2 regularization that avoids dominance of certain weights $w_{i,n}^L$ by penalizing high valued ones. It is a common strategy in deep learning to help train and avoid over-fitting by forcing a homogeneous weights distribution. It is weighted by the parameter λ_2 , which has been empirically set to $\lambda_2 = 0.001$ through trial-and-error. But the sensitivity to λ_2 is rather weak: different values have been tested, and when λ_2 is in the range $[0.01, 0.001]$, the results are nearly identical to those presented.

3.4.2 Neural network architecture and optimization

The neural network requires the dimensionless wall distance y^+ and the dimensionless pressure gradient p^+ as input, producing the dimensionless velocity u^+ as output. This fixes the structure of input and output layers in the neural network, which are respectively



(a) Probability density function ρ_i of the sample density.

(b) the loss weighting coefficient $w_{\rho_i} = 1/\rho_i$ used during neural network training.

Figure 3.6: Scatter of the training samples for iterative data-driven model. Probability density function ρ_i of the sample density and the loss weighting coefficient $w_{\rho_i} = 1/\rho_i$ used during neural network training.

composed by two and one neurons. Since the dynamical range of each input strongly differs (e.g., $y^+ \in [0, 10^2]$ and $p^+ \in [-0.02, 2]$), an additional hidden layer is added after the input layer to allow the NN to normalize the input values, as introduced in section 2.1.2.2.1.

In the context of wall-modeled RANS simulations, the neural network architecture (number of nodes and hidden layers) strongly impacts the CFD solver's CPU cost. For this reason, the neural network structure has been optimized to minimize the number of operations without compromising the accuracy of the prediction.

3.4.2.1 Optimization of the neural network architecture

An optimization process has been carried out to find optimal network architectures with respect to the accuracy/cost trade-off, in order to identify the optimal depth of the network and the ideal width of each layer.

A given network architecture with L_h hidden layers containing each a rather large number of neurons N_L is considered as starting point, and an optimization process which deactivates the least useful neurons is defined, which effectively consist in a structured pruning optimization, introduced in section 1.5.2.4. This yields a network architecture where the layer width is not constant anymore and has been reduced with minimal impact on the accuracy. The process has been repeated for several numbers of layers L_h , and is described below.

An optimization network is created by adding a gate directly after every hidden layer neuron. These additional nodes multiplies their input by a scalar (no activation function and no bias) which is a trainable parameter of the network. These gates are then densely connected to the downstream hidden layer. Therefore, suppressing a neuron from the network is equivalent to setting its gate's weight to zero. A schematic diagram of such a gated neural network is given in figure 3.7.

The initial network, composed of L_h layers and 16 neurons each, is first trained without gates, following the procedure described in 2.1.2.2.4 to minimize the loss ϵ defined in equation (3.21). Then, gates are added after all hidden-layer neurons and their weight is initialized to 1. The training is launched again (by keeping the same optimization

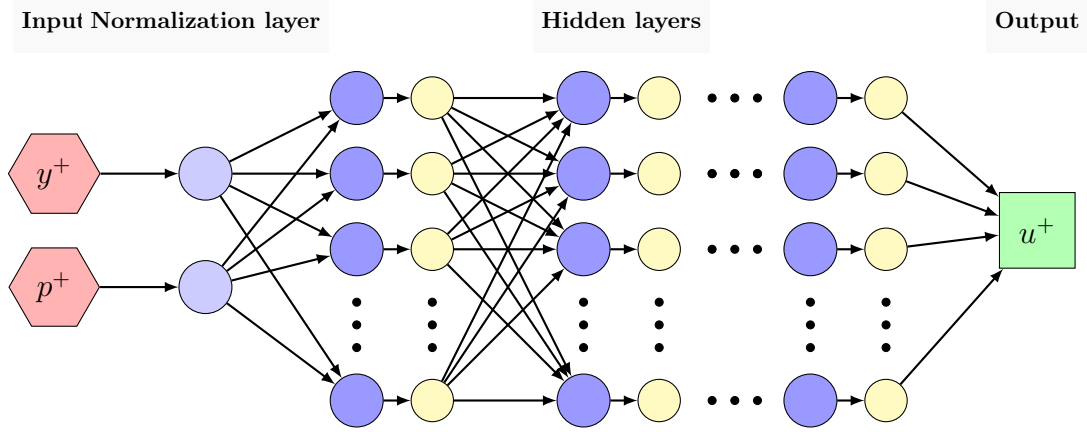


Figure 3.7: Schematic diagram of the feedforward neural network with multiple hidden layers used for structure optimization process.

parameters) with the following modified loss ϵ' :

$$\epsilon' = \epsilon + \frac{\lambda_1}{N_S} \|w_o^L\|_1, \quad (3.22)$$

where w_o^L is a vector containing all the weights of the gates and N_S the number of training samples. In the equation, ϵ refers to the loss function in (3.21). The extra L1 regularization term promotes sparsity and pushes toward setting some gates to zero. During the optimization, when a gate's weight drops below 0.01, it is permanently set to zero. Other threshold values, ranging from 0.01 to 0.1 have been tested. The highest values have been found to significantly affect the NN's accuracy. The penalization coefficient λ_1 controls the cost/accuracy trade-off by balancing the sparsity penalization with the rest of the loss (that promotes accuracy). Different values between 0.001 and 1 have been tested (the higher λ_1 , the more deactivated neurons) for different network depths L_h (2, 3, and 4). Once the optimization is converged, one gets the width of each layer. The resulting architecture is then evaluated through the Mean Absolute Percentage Error (MAPE).

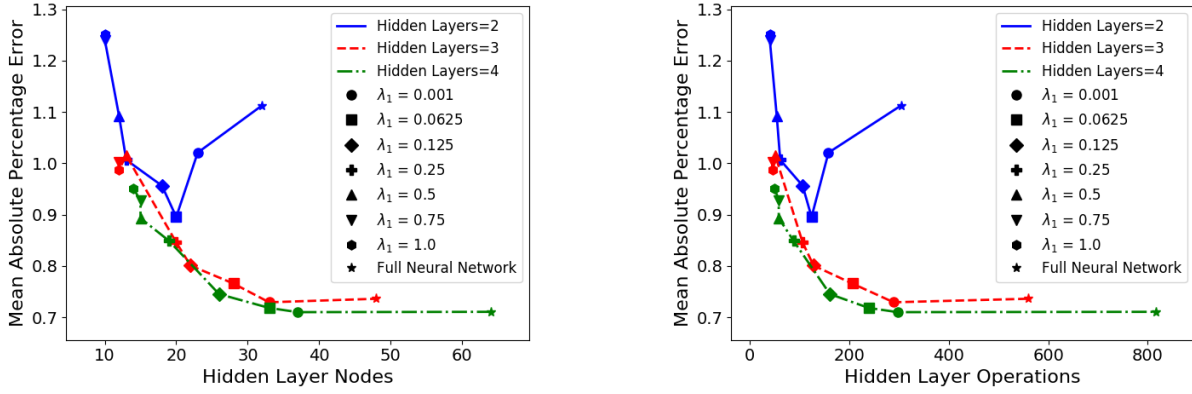
The error is thus monitored against the number of remaining hidden layer neurons and the number of operations in hidden layers (an operation here is considered as a link between two neurons, which corresponds to computing a quantity of the form $w_{i,j}^{(L)} O_j^{(L-1)}$, see equation (1.71). This provides Pareto fronts, which are shown in figure 3.8.

Overall, the three and four hidden layers neural networks showed similar levels of accuracy, while two hidden layer networks displayed significantly lower performances. As expected, a too low number of remaining nodes results in a step drop of accuracy.

The four hidden layer neural network trained with an L1 penalization coefficient of $\lambda_1 = 0.001$ (green circle) is retained as final architecture, motivated by a willing to favor accuracy over cost for this first methodological application. This neural network, composed by 10, 10, 10 and 7 nodes in the hidden layers, allows to more than halve the number of operations performed by the starting network given by four hidden layers of 16 nodes each while maintaining the similar level of accuracy (all lighter architectures yield a decreased accuracy).

3.4.2.2 Neural network architecture

The resulting optimized structure of the neural network consist in four main hidden layers, made of by 10, 10, 10 and 7 neurons, respectively. A schematic diagram of the NN



(a) MAPE as function of hidden layers nodes.

(b) MAPE as function of hidden layers operations

Figure 3.8: Results of structure optimization process. Mean Absolute Percentage Error (MAPE) given by neural networks with reference to the whole dataset.

architecture is given in figure 3.9.

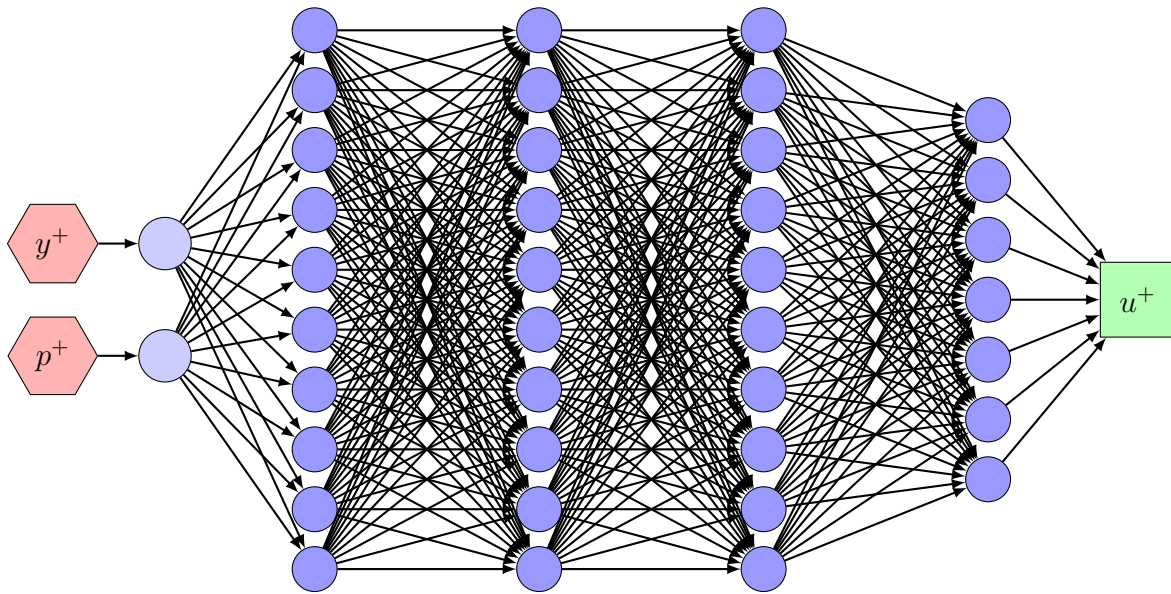
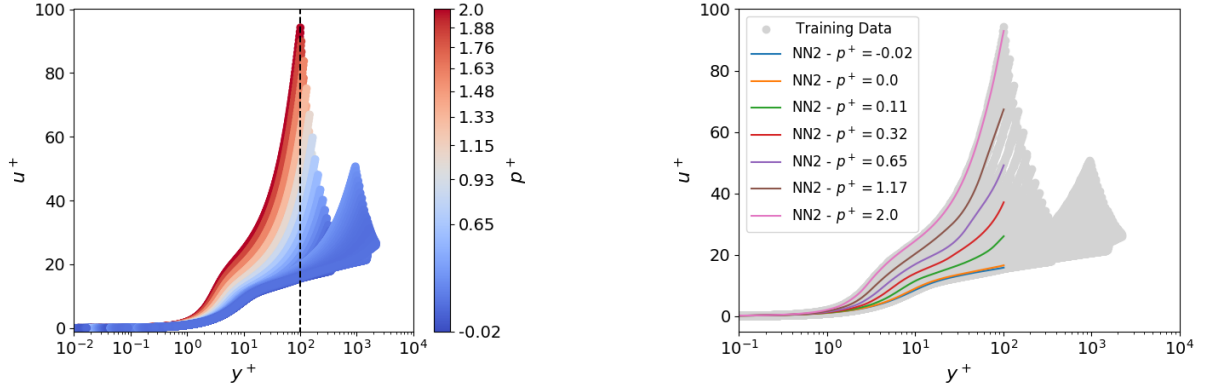


Figure 3.9: Schematic diagram of the resulting optimized feedforward neural network. Iterative data-driven model.

3.4.3 Training and a priori results

The training of the neural network is then carried out using a gradient-based algorithm, in our case, the Adam algorithm [59]. Overfitting is monitored by splitting data into a training and a validation dataset. The training dataset is used to update the parameters of the network during the optimization. The validation dataset is only used to evaluate the so-called validation loss. Overfitting can then be diagnosed if the validation loss significantly departs from the training loss. The training dataset is obtained through a random selection of 85% of data, while the validation data consist of the remaining 15%. The training of the neural network required 2004 epochs. More details on the neural



(a) Representation of samples with the inner region of the boundary layer ($\frac{y}{\delta} \leq 0.15$). The vertical dashed-line indicates the limit of the learning dataset $y^+ \leq 100$.

(b) Learned approximation of dimensionless wall velocity $u^+ = f(y^+, p^+)$. Comparison with training dataset.

Figure 3.10: Wall distance and pressure gradient informed neural network. Selected training data and learned wall normal evolution of the dimensionless velocity u^+ .

network training strategy are given in 2.1.2.2.4. The evolution of training and validation loss during the training process is given in figure 3.11.

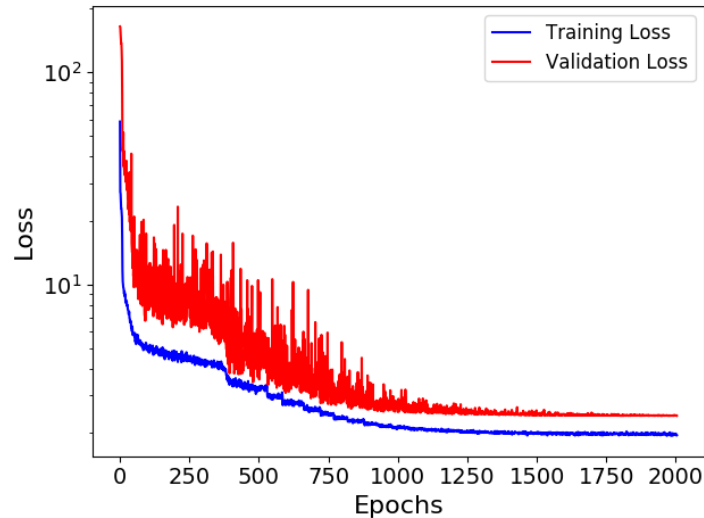


Figure 3.11: Training and validation loss evolution during training process.

The resulting learned relation $u^+ = f(y^+, p^+)$ is shown in figure 3.10, which also displays the training RANS dataset (only values below $y^+ \leq 100$ have been considered). An additional representation in the (y^+, p^+) -space may be seen in figure 3.12. The learned evolution of the dimensionless velocity u^+ expressed by the NN well reproduces the training dataset for the considered pressure gradients. Note that due to the geometric symmetry of the bump and its low height, the value range of the (dimensional) pressure gradient $\partial p / \partial x$ obtained in a given simulation is approximately symmetric (same maximal amplitude for positive and negative values). However, since the boundary layer flow is not symmetric with respect to the geometry (the friction velocity u_τ in particular), the range of dimensionless values p^+ is not symmetric, explaining why training data contains larger positive p^+ values than negative ones.

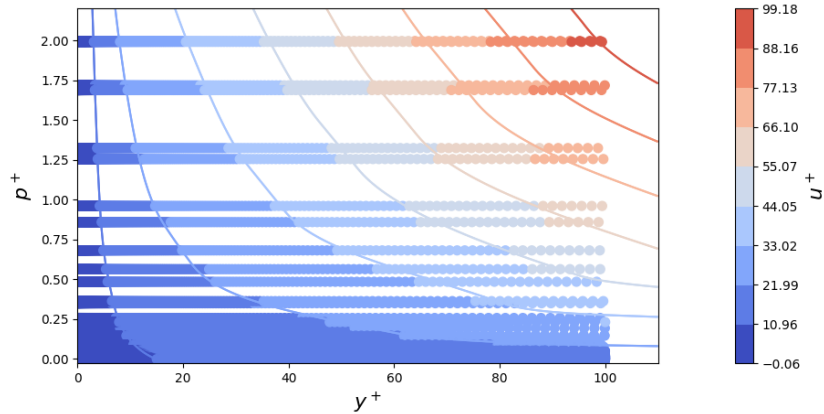


Figure 3.12: Contours of learned relation $u^+ = f(y^+, p^+)$ (colored iso-lines) and training dataset points (filled colored circles). The relation beyond $y^+ \approx 100$ and $p^+ \approx 2$ is obtained through linear extrapolation.

As seen in figure 3.12, the learning samples roughly cover $p^+ \in [-0.02, 2]$ and $y^+ \in [0, 100]$. To extend the capabilities of the model, if values beyond this range are encountered, then the neural network is replaced by a simple linear extrapolation based on $\partial f_{NN}/\partial y^+$ and $\partial f_{NN}/\partial p^+$ evaluated on the borders of the training domain, i.e., $y^+ \approx 100$ and $p^+ \approx 2$. This extrapolation is visible in figure 3.12.

3.5 Results

3.5.1 Test procedure

This section presents the results obtained using the neural network as a wall model, tested on various flow configurations and geometries defined in section 3.3. The results are compared with the wall-resolved RANS simulation for the same parameters.

The wall model is only applied to established turbulent boundary layers. For this reason, the complete simulation domain for the wall-modeled computation is limited to $X \in [0.3, 1.5]$, with an inflow condition that injects the fully developed boundary layer computed from the reference simulation at $X = 0.3$. Wall model performances are evaluated on the bump geometry only, from $X = 0.3$ to $X = 1.2$, which corresponds to the extraction zone for the training data.

The wall model is applied to structured grids, as explained in section 2.2. The numerical stencil of the spatial scheme in the solver includes two cell layers below the RANS - Wall model interface (R-WM I.), placed to a chosen value of y^+ . Therefore, two layers of cells below the interface are modeled. Figure 2.8a illustrate the encountered modeling configuration, while the case of model application illustrated in figure 2.8b is not found here, since the simulation domain is limited to $X \in [0.3, 1.5]$. The neglected cells are beyond the numerical stencil of the RANS region and are consequently unused during the RANS computation. The standard RANS integration takes over above the R-WM I. and the sampling point is taken at the second cell in the RANS integrated zone from the interface.

For each configuration, three y^+ values are considered for the R-WM I.: $y^+ \approx 10, 30, 50$. The first RANS-integrated cell is thus placed in the buffer layer, between the buffer layer and the logarithmic region, and in the logarithmic region, respectively.

The following subsections show first the global error for all the covered configurations, then more detailed results are presented and discussed for some selected cases.

3.5.2 Global errors

To evaluate the capabilities of the model, the wall model is tested on the ten test configurations proposed in figure 3.2, but also on the four training flows to obtain reference errors. The evaluation is thus performed both on seen and unseen configurations. Evaluating the model on all configurations allows comparing the error due to the approximate learned relation for u^+ and the impact of the assumptions made on other variables (temperature, density, and eddy viscosity, see section 3.1).

The evaluation of the global performances of the wall model is based on the estimation of the skin friction coefficient C_f on under-resolved grids compared to fully-resolved RANS simulations. A 2-norm error is computed between the reference RANS results and the simulation with the wall model. The global 2-norm error e_2 is obtained as

$$e_2 = \frac{\|C_f(X_i) - C_{f,ref}(X_i)\|_2}{\|C_{f,ref}(X_i)\|_2} = \frac{\sqrt{\sum_i [C_f(X_i) - C_{f,ref}(X_i)]^2}}{\sqrt{\sum_i [C_{f,ref}(X_i)]^2}} \quad (3.23)$$

where X_i are all the streamwise locations for X between 0.3 and 1.2.

Table 3.3 shows the global error e_2 computed for all the flow configurations and the three interface positions considered. The model shows good performances overall, with an error of 6.84% at most.

The global error for the training configurations (i.e. $h = 0.05 - Re = 10^6$, $h = 0.05 - Re = 6 \cdot 10^6$, $h = 0.07 - Re = 10^6$ and $h = 0.07 - Re = 6 \cdot 10^6$) is close to the error for unseen configurations. This shows that the learned relation does not suffer from interpolation or extrapolation issues in the range of flow conditions considered. However, the error increases as the RANS - Wall model interface is further away from the wall, and it appears to reduce when lower p^+ values are encountered, both for a lower bump height and an increase in the Reynolds number. This observation provides information on the main source of error in the present model, and it is further developed in section 3.5.3.

The simulation case ($h = 0.08$, $Re = 3 \cdot 10^6$) fails to converge when the RANS - Wall model interface is close to the wall (i.e. $y^+ \approx 10$). This configuration requires very high values of p^+ , well beyond the extrapolation capabilities of the proposed neural network (two orders of magnitude higher than the highest value encountered during training). Yet, the problem does not persist for cases with a higher interface (i.e., $y^+ = 30$ and $y^+ = 50$). This is due to an overestimate of the skin friction coefficient C_f and skin friction velocity u_τ , drastically reducing the sensed value of p^+ fed to the neural network. The cause of this overestimation of friction in high p^+ valued areas is addressed in section 3.5.4.

Nonetheless, this unconverged case is interesting because it bounds the extrapolation capabilities of the model. It appears that it may be unable to treat (quasi)separated boundary layers due to the attached nature of the flows considered for training. This limitation, as well as possible solutions, are further discussed in conclusion.

3.5.3 Interpolation test results

Interpolation capabilities of the model (evaluated on the interpolation test configurations defined in section 3.3) are presented in more detail in this section. The following results compare the obtained profiles for different flow variables from the wall models. It provides a more detailed view of the modeling errors. For conciseness, let us focus on the case

RANS interface at $y^+ \approx 10$					
	$h = 0$	$h = 0.05$	$h = 0.06$	$h = 0.07$	$h = 0.08$
$Re = 1 \cdot 10^7$			0.72%		0.88%
$Re = 8 \cdot 10^6$			0.75%		
$Re = 6 \cdot 10^6$		0.77%	0.8%	0.85%	
$Re = 3 \cdot 10^6$	0.61%	0.84%	0.93%	0.92%	*
$Re = 1 \cdot 10^6$		1.26%	1.35%	1.54%	
RANS interface at $y^+ \approx 30$					
	$h = 0$	$h = 0.05$	$h = 0.06$	$h = 0.07$	$h = 0.08$
$Re = 1 \cdot 10^7$			1.68%		2.09%
$Re = 8 \cdot 10^6$			1.92%		
$Re = 6 \cdot 10^6$		1.94%	2.27%	2.53%	
$Re = 3 \cdot 10^6$	1.21%	2.94%	3.41%	3.76%	3.98%
$Re = 1 \cdot 10^6$		5.54%	6.28%	6.84%	
RANS interface at $y^+ \approx 50$					
	$h = 0$	$h = 0.05$	$h = 0.06$	$h = 0.07$	$h = 0.08$
$Re = 1 \cdot 10^7$			2.9%		2.63%
$Re = 8 \cdot 10^6$			3.06%		
$Re = 6 \cdot 10^6$		3.05%	3.36%	3.54%	
$Re = 3 \cdot 10^6$	1.91%	3.57%	4.0%	4.34%	4.58%
$Re = 1 \cdot 10^6$		4.59%	5.1%	5.63%	

Table 3.3: 2-norm global error of the wall-modeled simulation on the skin friction coefficient C_f with respect to the reference wall-resolved RANS simulation for three different RANS interface positions and different combinations of bump height h and Reynolds number Re . The symbol * indicates that the computation failed to converge (non-convergence of the Newton-Raphson loop from algorithm 1)

$h = 0.06$ and $Re = 3 \cdot 10^6$ (results on the other interpolation cases are similar). It is referred to as an interpolation case because it involves a combination of Reynolds number and bump height inside the training range values.

Figure 3.13 shows the skin friction coefficient C_f along the wall and its error with respect to the wall-resolved simulation. The error e is normalized by the mean C_f value of the reference simulation over the wall, such that

$$e(X_i) = \frac{C_f(X_i) - C_{f,ref}(X_i)}{\overline{C_{f,ref}}}, \quad (3.24)$$

with X_i referring to the equally spaced solution points in X -coordinate direction along the considered portion of the wall, C_f the estimation of friction coefficient from the wall model, $C_{f,ref}$ the estimation of friction coefficient from the wall-resolved simulation and $\overline{C_{f,ref}}$ its mean value over the wall (the local value is not used for normalization to avoid artificial divergence of the error when C_f becomes too close to zero). Note that this error is purposely defined as a signed value (to evaluate potential model under/overestimation).

Globally, the wall model accurately estimates the skin friction evolution along the bump geometry, with the error increasing as the height of the interface increases. A maximum local error of 2%, 8.5% and 8.6% is found for respective y^+ values of 10, 30 and 50. The maximum error is found near the top of the bump or near its downstream bottom area (where high p^+ values are expected). Note that since the average value of C_f is used to normalize the error, it is expected to find larger errors where C_f reaches its maximal value.

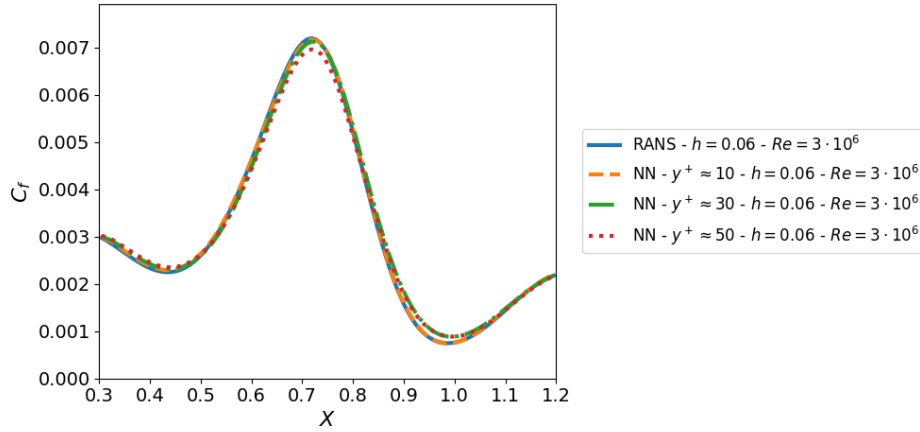
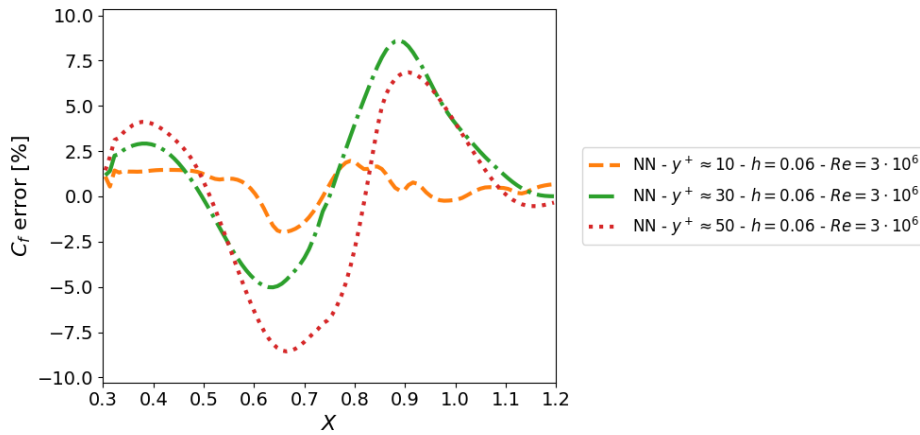
(a) Skin friction coefficient C_f (b) Normalized error e of skin friction coefficient C_f

Figure 3.13: Bump interpolation case ($h = 0.06$ and $Re = 3 \cdot 10^6$). Skin friction coefficient C_f along the X -coordinate direction and its normalized error with respect to the wall-resolved RANS simulation. Wall distance at first RANS computed cell: $y^+ \approx 10$, $y^+ \approx 30$, $y^+ \approx 50$.

Figure 3.14 shows tangential velocity profiles obtained at $X = 0.75$ (top of the bump) and at $X = 0.95$ (downstream bottom area). The curves present the results of the wall model for the three considered RANS - Wall model interface positions to the reference wall-resolved RANS simulation. At the top of the bump, the model closely matches fully resolved RANS results. In the downstream bottom area, the model still fits the RANS simulation when the R-WM I. is located at $y^+ \approx 10$. The error increases for simulations with higher interfaces with a tendency to overestimate the velocity. Simulation with an interface at $y^+ \approx 30$ and $y^+ \approx 50$ show a very similar error and behavior.

The temperature evolution from the wall at $X = 0.75$ and $X = 0.95$ are shown in figure 3.15. For brevity, density profiles are omitted here since their behavior closely reproduces those of the temperature. Again, the profiles depart further from the reference as the interface height increases. Nonetheless, for all simulations, the relative error on the temperature is very limited (below 1%) compared to the wall-resolved simulation.

Figure 3.16 shows the evolution of the dimensionless velocity u^+ with the wall distance y^+ obtained from the wall model at $X = 0.75$ and $X = 0.95$. The lower end of the curve (near-wall region) is fixed by the wall model, while the upper behavior is driven by the S.-A. RANS integration based on the values of the modeled region. The near-wall area shows

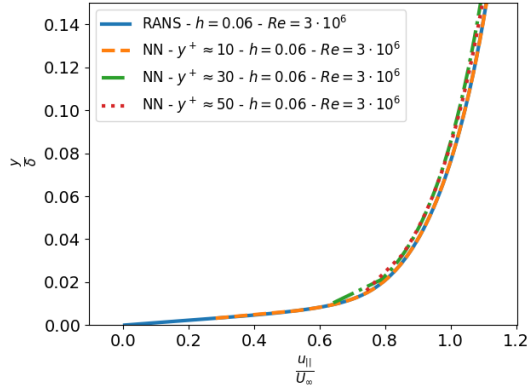
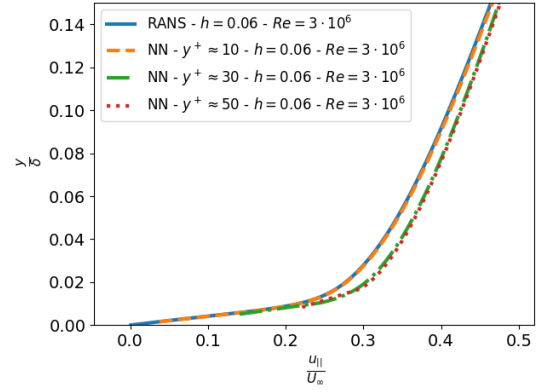
(a) $X = 0.75$ (top of the bump)(b) $X = 0.95$

Figure 3.14: Bump interpolation case ($h = 0.06$ and $Re = 3 \cdot 10^6$). Wall normal evolution of tangential velocity at $X = 0.75$ (top of the bump) and at $X = 0.95$. Wall distance at first RANS computed cell: $y^+ \approx 10$, $y^+ \approx 30$, $y^+ \approx 50$.

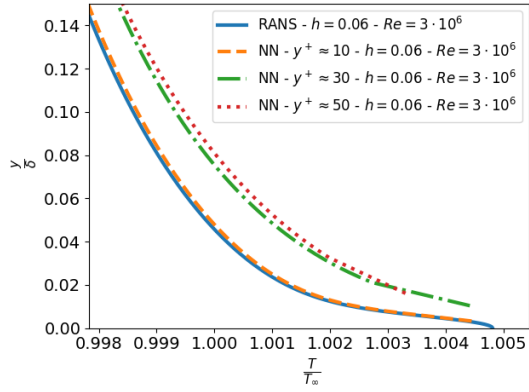
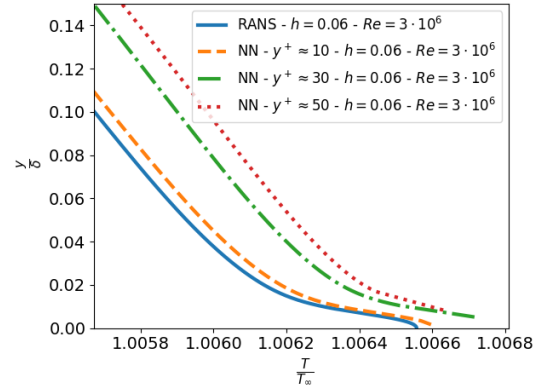
(a) $X = 0.75$ (top of the bump)(b) $X = 0.95$

Figure 3.15: Bump interpolation case ($h = 0.06$ and $Re = 3 \cdot 10^6$). Wall normal evolution of temperature at $X = 0.75$ (top of the bump) and at $X = 0.95$. Wall distance at first RANS computed cell: $y^+ \approx 10$, $y^+ \approx 30$, $y^+ \approx 50$.

a close match with respect to the reference RANS simulation. However, the integration in the RANS region yields a more significant error that increases with the wall distance. Overall, the profiles display a good agreement with the reference solution at $X = 0.75$, while only the wall model with an interface at $y^+ \approx 10$ well agrees with the RANS results in the downstream bottom area ($X = 0.95$). Again, models with $y^+ \approx 30$ and $y^+ \approx 50$ interfaces show similar behavior.

Figure 3.17 shows the dimensionless S.-A. variable \tilde{v}^+ at $X = 0.75$ and $X = 0.95$. Again, the errors increase with higher wall distances at the first cell. The overall discrepancy appears limited at the top of the bump, while differences are found more relevant in the downstream bottom area of the bump. These differences explain the error behavior linked to the wall model strategy. The downstream bottom part of the bump displays a high p^+ value, which appears to strongly influence the dimensionless S.-A. variable \tilde{v}^+ compared to the modeled one (dashed line). The S.-A. variable in the presence of strong adverse pressure gradients tends to depart from a linear behavior for lower y^+ values, which explains the growing error observed for higher RANS - Wall model interface (other

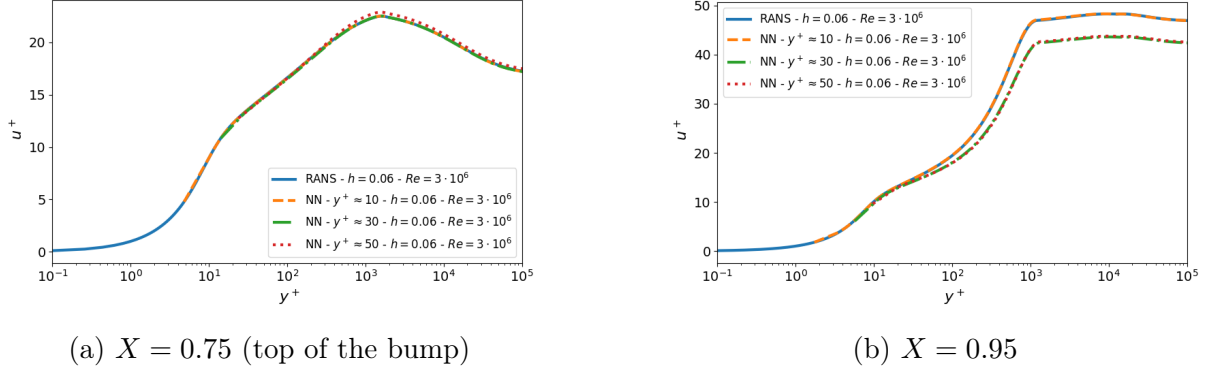


Figure 3.16: Bump interpolation case ($h = 0.06$ and $Re = 3 \cdot 10^6$). Wall normal evolution of dimensionless velocity at $X = 0.75$ (top of the bump) and at $X = 0.95$. Wall distance at first RANS computed cell: $y^+ \approx 10$, $y^+ \approx 30$, $y^+ \approx 50$.

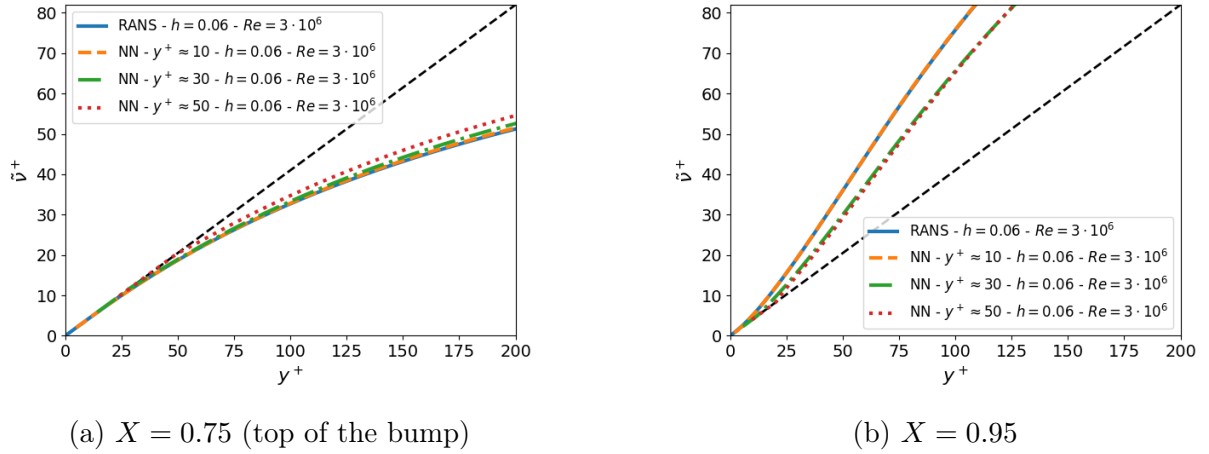


Figure 3.17: Bump interpolation case ($h = 0.06$ and $Re = 3 \cdot 10^6$). Wall normal evolution of dimensionless S.-A. variable \tilde{v}^+ at $X = 0.75$ (top of the bump) and at $X = 0.95$. Wall distance at first RANS computed cell: $y^+ \approx 10$, $y^+ \approx 30$, $y^+ \approx 50$.

modeling assumptions or approximations do not have an increasing error for higher y^+). This also explains why the global error from table 3.3 tends to increase for lower Re and higher h : higher p^+ values are expected with lower Reynolds numbers and higher bump heights, leading to more significant errors on \tilde{v}^+ .

The error on the S.-A. variable impacts the RANS integration above the model interface. It strongly affects flow quantities sensed at the sample point. For this reason, a high error on \tilde{v}^+ leads to underestimated p^+ values at the sample point. Thus, the wall model converges toward overestimated solutions for the skin friction coefficient. The opposite situation happens at the top of the bump, where the poor modeling of \tilde{v}^+ for higher interface height yields underestimated C_f . Additionally, for lower Reynolds number configurations, the dimensionless S.-A. \tilde{v}^+ variable departs more rapidly from a linear behavior, which explains the observed error differences for different values of Re in Table 3.3 as the interface is moved upward.

3.5.4 Extrapolation cases

Extrapolation capabilities of the model have been tested on unseen configurations during the training process with Reynolds number Re and a bump height h combinations beyond

the range of values met in the training dataset. For almost all tested configurations, the error and conclusions are similar to those from the interpolation cases: the error on the C_f evolution, velocity, temperature, density, and eddy viscosity profiles is the highest near the top of the bump and its downstream bottom area, with the same tendencies to over/underestimate the flow variables as previously. The performance assessment on the particular case of a zero-pressure flat plate ($h = 0$) is covered in section 3.5.4.1. For $h \neq 0$, the error becomes significant and may lead to convergence problems when the flow is close to separation. This is discussed in more detail in section 3.5.4.2.

3.5.4.1 Flat plate case

The extrapolation capabilities of the model have been tested on the flat plate flow at $Re = 3 \cdot 10^6$. Figure 3.18 shows respectively the predicted skin friction coefficient C_f and its relative error e with respect to wall-resolved RANS (the normalizing value is the averaged value of C_f)

The model appears to be well adapted to quasi-equilibrium boundary layer, since the skin friction coefficient is well reproduced by the neural network. The local normalized error does not exceed 3% even when the model interface is located at $y^+ \approx 50$; yet, better performances are achieved when the interface is located closer to the wall.

3.5.4.2 Near separation case

This section focuses on the case $h = 0.08$ and $Re = 3 \cdot 10^6$. This is the test case with the strongest adverse pressure gradient, the flow being on the verge of separation. Thus, it is the most challenging case for our wall model, which was not designed to handle separated regions. Convergence problems appeared when the RANS interface was close to the wall, as reported in table 3.3. Figure 3.19 shows the results obtained for the cases that were able to converge (interface at $y^+ \approx 30$ and $y^+ \approx 50$). One may see that the evolution of C_f is qualitatively good. However, the error curve shows that the behavior is slightly erratic near the point where C_f approaches zero. Additionally, the figure shows the model's tendency to overestimate C_f in this region. As in section 3.5.3, this overestimation becomes less significant when the interface is closer to the wall. That explains why only the case with the interface at $y^+ \approx 10$ failed to converge: the modeling error on $\tilde{\nu}^+$ is smaller. Thus the encountered values of p^+ become closer to the reference, i.e., too high to be handled properly by the model. These results show that the wall model may handle reasonably large values of p^+ (such as those encountered when the interface is higher, underestimated due to the S.-A. variable modeling), but it reaches its limit for the case $y^+ = 10$.

3.5.4.3 Influence of dimensionless pressure gradient

To assess the importance of including or not the dimensionless pressure gradient p^+ as an input parameter to the wall law $u^+ = f(y^+, p^+)$, the selected interpolation case of the bump flow with $h = 0.06$ and $Re = 3 \cdot 10^6$ has been evaluated by imposing $p^+ = 0$ in the previously learned wall law during the CFD computation, i.e., $u^+ = f(y^+, p^+ = 0)$. Figure 3.20 shows the normalized error on skin friction coefficient C_f comparing the neural network fed with the wall distance y^+ and the dimensionless pressure gradient p^+ and the same neural network solely fed with the wall distance y^+ . Overall, the pressure gradient-informed wall model manages to reproduce wall-resolved RANS simulation better. Significant differences between neural networks are detected at the top of the bump (i.e.,

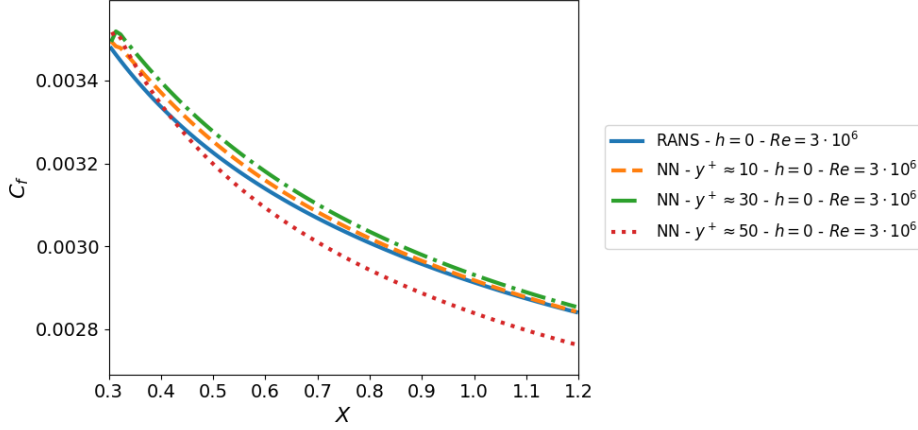
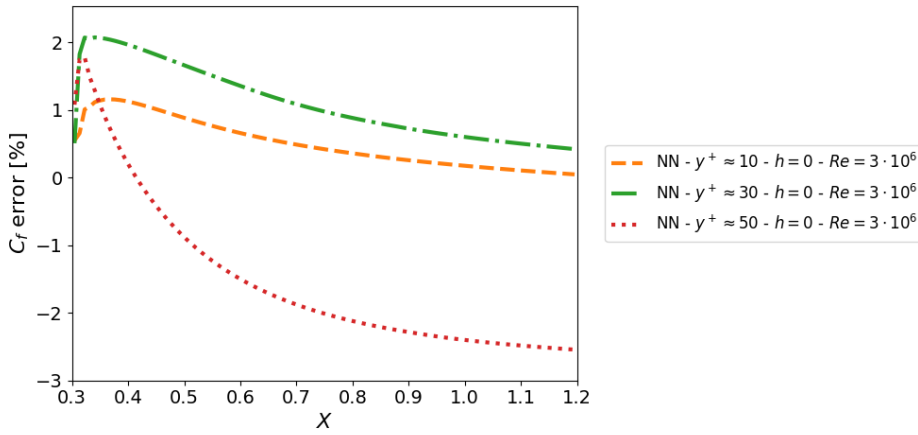
(a) Skin friction coefficient C_f (b) Normalized error of skin friction coefficient C_f

Figure 3.18: Bump interpolation case ($h = 0.06$ and $Re = 3 \cdot 10^6$). Wall normal evolution of temperature at $X = 0.75$ (top of the bump) and at $X = 0.95$. Wall distance at first RANS computed cell: $y^+ \approx 10$, $y^+ \approx 30$, $y^+ \approx 50$.

$X = 0.75$) and downstream of the bump geometry (i.e., $X = 0.95$), where the highest pressure gradients are expected.

Table 3.4 shows 2-norm global errors on the skin friction coefficient C_f computed with wall distance and pressure gradient informed neural network $u^+ = f(y^+, p^+)$, and solely wall distance-informed neural network $u^+ = f(y^+, p^+ = 0)$. The 2-norm error of wall distance and pressure gradient informed neural network is extracted from table 3.3. The capabilities of the pressure gradient-fed neural network to estimate better local skin friction coefficient C_f is also confirmed on global 2-norm errors.

3.5.5 Mass conservation

As mentioned earlier, the present wall model strategy does not enforce the conservation of mass, which may be problematic for internal flow simulations. The mass loss in our configurations is evaluated by integrating the mass flux over the limits of the computational domain Ω (i.e., $X = 0.3$, $X = 1.2$ and $Y = 5$) by excluding the lower wall where the wall law is applied (since no mass flux exists there):

$$\oint_{\Omega} \rho \mathbf{u} \cdot \mathbf{n} d\Omega, \quad (3.25)$$

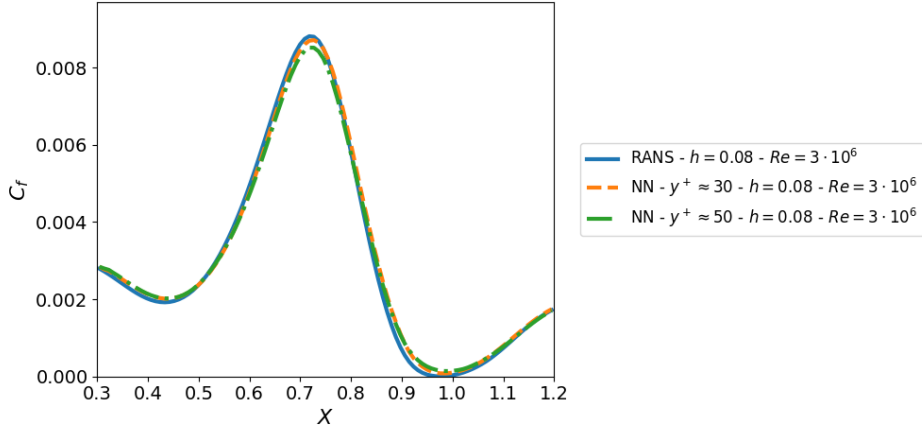
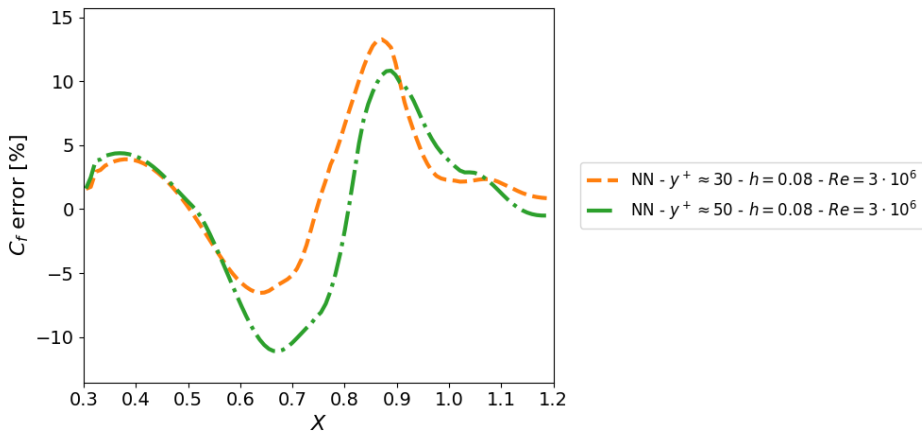
(a) Skin friction coefficient C_f (b) Normalized error e of skin friction coefficient C_f

Figure 3.19: Near separation case ($h = 0.08$ and $Re = 3 \cdot 10^6$). Skin friction coefficient C_f along X -coordinate direction and its normalized error with reference to wall-resolved RANS simulation. Wall distance at first RANS computed cell: $y^+ \approx 50$.

	$h = 0.06 - Re = 3 \cdot 10^6$	
	$u^+ = f(y^+, p^+)$	$u^+ = f(y^+, p^+ = 0)$
$y^+ \approx 10$	2.4%	3.04%
$y^+ \approx 30$	3.69%	4.57%
$y^+ \approx 50$	4.0%	6.41%

Table 3.4: Relative 2-norm global error on the skin friction coefficient C_f of the wall modeled simulation with respect to the reference wall-resolved RANS simulation. Bump interpolation case ($h = 0.06$ and $Re = 3 \cdot 10^6$) for three different RANS interface positions. Comparison between the wall distance and pressure gradient informed neural network $u^+ = f(y^+, p^+)$ and solely wall distance informed neural network $u^+ = f(y^+)$. 2-norm error of wall distance and pressure gradient informed neural network is extracted from table 3.3

with \mathbf{n} the exterior normal to the contour. This quantity is supposed to be null, and mass non-conservativity is characterized by the ratio

$$\frac{\oint_{\Omega} \rho \mathbf{u} \cdot \mathbf{n} d\Omega}{\rho_{\infty} U_{\infty} \delta}. \quad (3.26)$$

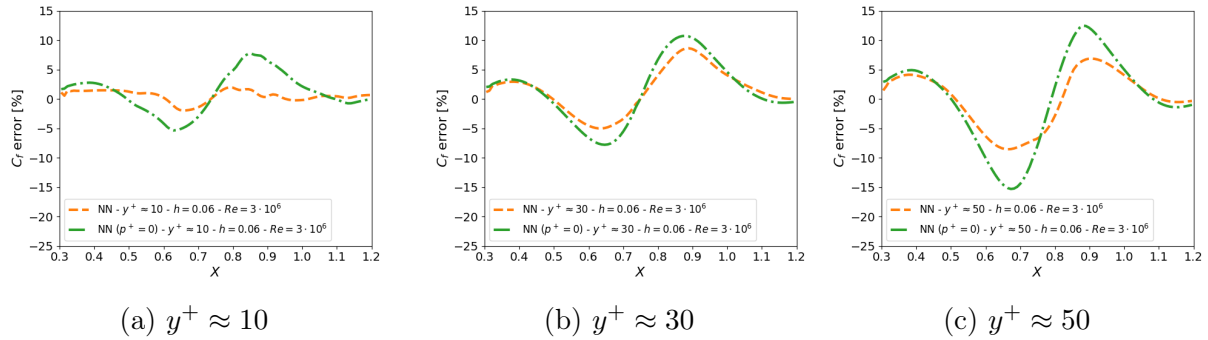


Figure 3.20: Bump interpolation case ($h = 0.06$ and $Re = 3 \cdot 10^6$). Normalized error e of skin friction coefficient C_f obtained by solely wall distance informed neural network ($u^+ = f(y^+, p^+ = 0)$). Wall distance at first RANS computed cell: $y^+ \approx 10$, $y^+ \approx 30$, $y^+ \approx 50$.

The lost mass rate is normalized with respect to the free-stream flow rate entering a section of height δ , which corresponds to the boundary layer thickness at the beginning of the evaluation zone (i.e., $X = 0.3$). The maximum loss equals 0.44% for the case $h = 0.07$ and $Re = 10^7$ with the RANS - Wall model interface at $y^+ \approx 50$. For external aerodynamics, such an error may be acceptable; for internal flows, where the mass-flow rate may be an important quantity, such an error might become problematic.

3.6 Conclusion

This chapter presents a new deep learning-based approach to wall models for RANS simulations inspired by classical wall laws. The proposed wall models rely on wall dimensionless quantities, here the wall distance y^+ and the wall pressure gradient p^+ , to reconstruct the dimensionless wall velocity u^+ profiles in wall-bounded region.

The model provides embedded neural networks to the CFD solver code, which forces the primitive variables in modeled cells at a given interface near the wall, below which the RANS computation is disabled. It is equivalent to a Dirichlet boundary condition applied to the conventional RANS region.

The deep learning-based approach consists of a wall distance and pressure gradient-informed neural network trained on a dataset extracted from a fine wall-resolved RANS simulation of the flow over a bump. The training process has been performed with different Reynolds number conditions and pressure gradient levels based on the bump height.

The neural network has been tested and compared with a fully resolved RANS simulation. The test cases were selected both from the training dataset and unseen configurations of the bump flow, characterized by a different combination of Reynolds number and bump height. The particular case of a flat plate was also included for testing. The benchmark cases have been run with varying interface heights to test the proposed wall model with various wall distances of the RANS interface.

One test case was particularly challenging. Convergence issues were found for nearly-separated cases with a strong adverse pressure gradient. This was expected, since no particular treatment has been designed to enable the network to handle such cases. This may be addressed in the future by including a significant number of nearly-separated and separated cases in the training database.

The wall distance and pressure gradient informed network yields accurate results for almost all the test cases and modeling distances. However, the model underestimates the skin friction coefficient, with an error that increases as the interface height becomes higher. This is mainly due to the increasing deviation of the Spalart-Allmaras variable behavior from the linear modeling approach as the distance from the wall and pressure gradients increase. Consequently, further considerations are needed to extend the validity domain of the wall model to greater modeling distances. For instance, more advanced treatments to impose the near wall behavior of the turbulence model. The modeling of the Spalart-Allmaras variable in the present case or the wall normal velocity component could be explored.

Another known shortcoming is that the proposed methodology could lead to conservativity problems, even though it was found to be negligible in the present applications.

Even though relatively simple geometries characterize the test cases, this chapter highlights the potential of neural networks for wall-bounded region modeling. In particular, searching a relation between non-dimensional quantities mechanically gives the network some extrapolation capabilities, enabling, for instance, simulation for Reynolds number beyond the range considered for training. This would not be possible if one tries to estimate dimensional quantities.

Moreover, even if a methodology has been proposed to optimize the computational cost of the network, the iterative estimation of wall shear stress is not well adapted to reducing computational cost, as it requires multiple neural network inferences. This question may require more extensive attention and future work, since it is a critical point in the context of wall models for CFD.

The present results are the starting point for further studies and investigations required to overcome the issues encountered during this chapter.

Chapter 4

Efficient data-driven wall models for RANS

4.1	Wall law formulation	91
4.1.1	Dirichlet-To-Neumann formulation for the wall tangent velocity evolution	91
4.1.1.1	Additional parameters for Dirichlet-To-Neumann map	92
4.1.2	Physical model for thermodynamic state and wall normal velocity field	92
4.1.3	Near-wall Spalart-Allmaras modeling	93
4.2	Numerical implementation of the wall model	94
4.2.1	Wall model discretization	94
4.2.2	Wall model application	94
4.2.3	Numerical effect of Dirichlet-To-Neumann approach	95
4.3	Flow configurations	97
4.3.1	Bump flow case	97
4.3.1.1	Training and testing datasets	97
4.3.2	Airfoil case	98
4.4	Data-driven modeling	98
4.4.1	Neural networks	98
4.4.2	Neural network architecture	99
4.4.3	Dataset treatment	99
4.4.4	Loss function selection	101
4.4.5	Training strategy	102
4.4.6	Discussion on direct estimation and derivative computation	102
4.4.7	Discussion on single and multiple neural networks approach	103
4.5	Results	105
4.5.1	Results on bump geometry	105
4.5.1.1	Test procedure	105
4.5.1.2	Assessment of Spalart-Allmaras modeling strategy	107
4.5.1.3	Discussion on the additional parameters for DtN-map	108
4.5.1.4	Model validation and comparison with iterative approach	109
4.5.1.5	Interpolation in Reynolds number	112
4.5.1.6	Interpolation in bump height	112
4.5.1.7	Interpolation in bump height and Reynolds number	113
4.5.2	Test on the airfoil geometry	114
4.5.2.1	Test setup and procedure	115
4.5.2.2	Results	115
4.5.3	Numerical performances assessment	117

4.5.3.1	Model computational time assessment	117
4.5.3.2	Total convergence time assessment	120
4.6	Conclusion	121

In the previous chapter, it is demonstrated that the dependency of the wall-law with respect to the streamwise pressure gradient along the streamwise direction x could be learned from data by considering the additional non-dimensional parameter p^+ [1] in the wall-law:

$$u^+ = f(y^+, p^+), \quad \text{with} \quad u^+ = \frac{u_{\parallel}}{u_{\tau}}, \quad y^+ = \frac{u_{\tau}}{\nu_w} y \quad \text{and} \quad p^+ = \frac{\nu_w}{\rho_w u_{\tau}^3} \frac{\partial p}{\partial x}. \quad (4.1)$$

This generalized wall-law can be modeled by a neural network optimized to fit data obtained from wall-resolved simulations. The resulting data-based model has then been implemented within a RANS code and showed fairly good accuracy for close to wall modeling distances (i.e., below $y^+ \approx 50$).

Yet, their first approach displays several weaknesses. Firstly, the resulting model is computationally expensive due to the many neural network evaluations required for the iterative solution of the local skin friction, which is obtained by solving

$$\frac{u_{\parallel}^S}{u_{\tau}} = f\left(\frac{u_{\tau}}{\nu_w} y^S, \frac{\nu_w}{\rho_w u_{\tau}^3} \frac{\partial p}{\partial x}\right), \quad (4.2)$$

given by equation (4.1) when applied to a sampling point S

Secondly, the model shows poor accuracy when applied in the log-layer at large distances y^+ from the wall. In the approach of Romanelli et al. [92], presented in chapter 3, the turbulent variable behavior is approximated by a linear evolution of the dimensionless Spalart-Allmaras variable, defined as

$$\tilde{\nu}^+ = \frac{\rho \tilde{\nu}}{\mu}. \quad (4.3)$$

Following the study of Kalitzin et al. [56], the evolution of the dimensionless Spalart-Allmaras variable in the near-wall can be modeled as

$$\tilde{\nu}^+ = \kappa y^+, \quad (4.4)$$

where $\kappa = 0.41$ is the Von Kármán constant. This model is applicable for the inner region of low-pressure boundary layers, in the viscous and logarithmic layer. However, experiences documented in chapter 3 indicate that the model accuracy diminishes rapidly when encountering non-negligible pressure gradients, and the loss of accuracy increases with the modeling distance from the wall.

Lastly, the parameter p^+ is not sufficient to fully characterize the internal state of the boundary layer (the wall-resolved RANS simulations showed that multiple values of skin friction could arise for a given p^+ , see figure 3.4).

In the present chapter, to address the computationally expensive estimation of the iterative solution of the local skin friction, the wall-law is reformulated in the general framework of a Dirichlet-To-Neumann (DtN) map, which assumes the existence of a relation (a map) between the value of the streamwise velocity at some height of the boundary layer and its wall-normal derivative. This framework is very natural and elegant for the implementation of boundary conditions. This strategy avoids solving the implicit

equation determining the skin-friction, rendering the computation much faster. Then, the turbulence modeling is improved by removing the linear constraint of the pseudo eddy viscosity (which is only true very close to the wall). Overall, as shown below, the model is much faster and accurate over the considered parameter space. Finally, to reduce model inaccuracies, especially at large distances from the wall, additional input features are considered to better characterize the internal state of the boundary layer.

The content of this chapter, presented at the ECCOMAS 2024 conference, has been submitted to the journal "Journal of Computational Physics" and is currently under review.

4.1 Wall law formulation

As introduced in section 2.2, the proposed wall model aims to reconstruct the flow state in the near-wall subdomain Ω_M to transfer the no-slip boundary condition from a point P on the wall surface to an equivalent boundary condition at point I , located further from the wall along the normal direction from P . Refer to figure 2.5.

This section introduces the key relations used to reconstruct the flow state within the subdomain Ω_M .

4.1.1 Dirichlet-To-Neumann formulation for the wall tangent velocity evolution

The wall law in (4.1) may be differentiated to obtain the DtN formulation: let us introduce $g(y^+, p^+)$ such that

$$\frac{\partial u^+}{\partial y^+} = g(y^+, p^+), \quad (4.5)$$

where $g(y^+, p^+) = \frac{\partial}{\partial y^+} f(y^+, p^+)$. Then, the explicit definition of the variables u^+ , y^+ and p^+ in equation (4.1) shows that u_τ is an implicit function of the dimensional variables $(u, y, \nu_w, \rho_w, \frac{\partial p}{\partial x})$. By dimensional analysis, considering the velocity scale $LT^{-1} = \nu_w/y$ and length-scale $L = y$, there exists a function h such that:

$$y^+ = h(\eta, \beta) \quad \text{with} \quad \eta = \frac{u_\parallel y}{\nu_w}, \quad \beta = \frac{y^3}{\rho_w \nu_w^2} \frac{\partial p}{\partial x}. \quad (4.6)$$

The DtN map $u_\parallel \rightarrow \frac{\partial u_\parallel}{\partial y}$ may then be obtained from the knowledge of g and h . Knowing that

$$\frac{\partial u_\parallel}{\partial y} = \frac{u_\tau^2}{\nu_w} \frac{\partial u^+}{\partial y^+} \quad \text{with} \quad u_\tau = \frac{\nu_w}{y} y^+, \quad (4.7)$$

the map can be formulated as follows

$$\frac{\partial u_\parallel}{\partial y} = \frac{\nu_w}{y^2} h^2(\eta, \beta) g \left(h(\eta, \beta), \frac{\beta}{h^3(\eta, \beta)} \right) = \frac{\nu_w}{y^2} F(\eta, \beta). \quad (4.8)$$

Considering the figure 2.5, the formulation in equation (4.8) can be used to replace the wall boundary condition at a point P by a Neumann boundary condition on the surface Γ_I at a point I . Moreover, integrating equation (4.8) in the wall normal direction, starting from the interface point I allows to reconstruct the velocity profile entirely through Ω_M .

The next section introduce two additional parameters to characterize the state of the internal boundary layer and improving its modeling.

4.1.1.1 Additional parameters for Dirichlet-To-Neumann map

Section 4.1.1 introduced the methodology for a DtN strategy derived from the relation $\frac{\partial u^+}{\partial y^+} = g(y^+, p^+)$. However, for a better modeling of the wall tangent velocity profile, the proposed wall model includes two additional parameters to characterize the state of the internal boundary layer. These parameters are based on the streamwise second derivative of the pressure $\frac{\partial^2 p}{\partial x^2}$ and the Spalart-Allmaras variable $\tilde{\nu}$. They follow the same non dimensionalization process detailed in section 4.1.1. Further discussion on the additional parameters and the motivation for their selection is provided in 4.5.1.3.

Considering the additional parameters, the DtN map reads:

$$y^+ = h(\eta, \beta, \theta, \zeta) \quad \text{and} \quad \frac{\partial u^+}{\partial y^+} = g(y^+, p^+, \partial p^+, \tilde{\nu}^+) \quad (4.9)$$

respectively with

$$\eta = \frac{u_{\parallel} y}{\nu_w}, \quad \beta = \frac{y^3}{\rho_w \nu_w^2} \frac{\partial p}{\partial x}, \quad \theta = \frac{y^4}{\rho_w \nu_w^2} \frac{\partial^2 p}{\partial x^2}, \quad \zeta = \frac{\tilde{\nu}}{\nu_w} \quad (4.10)$$

and

$$y^+ = \frac{u_{\tau} y}{\nu_w}, \quad p^+ = \frac{\nu_w}{\rho_w u_{\tau}^3} \frac{\partial p}{\partial x}, \quad \partial p^+ = \frac{\nu_w^2}{\rho_w u_{\tau}^4} \frac{\partial^2 p}{\partial x^2}, \quad \tilde{\nu}^+ = \frac{\tilde{\nu}}{\nu_w}, \quad (4.11)$$

which are obtained by re-scaling the parameters set in equation (4.10), using the wall shear stress non dimensionalization, as shown below

$$y^+ = h(\eta, \beta, \theta, \zeta), \quad p^+ = \frac{\beta}{h^3(\eta, \beta, \theta, \zeta)}, \quad \partial p^+ = \frac{\theta}{h^4(\eta, \beta, \theta, \zeta)} \quad \text{and} \quad \tilde{\nu}^+ = \zeta. \quad (4.12)$$

4.1.2 Physical model for thermodynamic state and wall normal velocity field

The wall-normal temperature profiles in the Ω_W portion of the domain and along the Γ_I boundary (see figure 2.5) are again modeled using the Crocco-Busemann's relation, introduced in equation (3.8). The continuity of the temperature profile across Γ_I is ensured by estimating the unknown quantities in equation (3.8), solving the linear system in (3.9).

The pressure evolution is modeled by enforcing both a zero wall-normal pressure gradient ($\partial p / \partial y$) at the wall and the continuity condition at the interface point I . This leads to a polynomial modeling of the pressure

$$p(y) = P_2(y) = \sum_{k=0}^2 a_k y^k. \quad (4.13)$$

The coefficients a_k are obtained by solving the system below:

$$\left\{ \begin{array}{l} \frac{\partial}{\partial y} P_2(0) = 0 \\ P_2(y_I) = p_I \\ \frac{\partial}{\partial y} P_2(y_I) = \frac{\partial p}{\partial y} \Big|_{y=y_I} \end{array} \right\} p(y) \in C^1. \quad (4.14)$$

where I relates to the interface point.

The density ρ can thus be obtained using the perfect gas law, knowing the temperature and pressure evolution. Furthermore, with knowledge of the temperature T , the molecular viscosity μ is computed using the selected viscosity model. Sutherland's law is used in our case.

Moreover, the first and the second derivatives of the pressure field along the streamwise direction, respectively $\frac{\partial p}{\partial x}$ and $\frac{\partial^2 p}{\partial x^2}$ are computed at the sampling point S and are considered constant along the wall normal direction linking point P and S in figure 2.5.

Finally, as for the pressure evolution, the wall normal velocity u_\perp is obtained through a polynomial modeling. Specifically, a third-degree polynomial is employed, with coefficients chosen to ensure C^1 continuity of the velocity profile at the interface point I , as well as a zero velocity and a zero gradient in the wall-normal direction at the wall. The latter condition is given by the continuity equation for incompressible flows

$$\frac{\partial u_\parallel}{\partial x} + \frac{\partial u_\perp}{\partial y} = 0, \quad (4.15)$$

knowing that $\frac{\partial u_\parallel}{\partial x} \Big|_{y=0} = 0$.

4.1.3 Near-wall Spalart-Allmaras modeling

This section illustrates the approach used to reproduce the near-wall behavior of the turbulence model in the Ω_W portion of the domain and along the Γ_I boundary. Since the turbulence model employed is the Spalart-Allmaras model, the near-wall modeling is based on the evolution of the Spalart-Allmaras variable $\tilde{\nu}$.

The inaccuracy of the linear evolution model is addressed by replacing the modeling strategy. The behavior of the dimensionless Spalart-Allmaras viscosity is now reproduced by a polynomial regression. A third degree polynomial

$$\tilde{\nu}^+(y^+) = P_3(y^+) = \sum_{k=0}^3 a_k y^{+k} \quad (4.16)$$

is employed in the near-wall zone to estimate $\tilde{\nu}^+$ as function of the dimensionless wall distance y^+ . The coefficients a_k are obtained by solving the system below

$$\left\{ \begin{array}{l} P_3(0) = 0 \\ \frac{\partial}{\partial y^+} P_3(0) = \kappa \\ P_3(y_I^+) = \tilde{\nu}_I^+ \\ \frac{\partial}{\partial y^+} P_3(y_I^+) = \frac{\partial \tilde{\nu}^+}{\partial y^+} \Big|_{y^+=y_I^+} \end{array} \right\} \tilde{\nu}^+(y^+) \in C^1. \quad (4.17)$$

In the system (4.17), the first two equations set respectively the value and the wall normal derivative of the dimensionless Spalart-Allmaras variable at the wall. Here, the slope proposed by the linear evolution model is imposed, since the model demonstrated sufficient accuracy for low wall distance modeling, i.e. $y^+ \leq 10$, even in the presence of moderate pressure gradients. The last two equations are instead addressed to ensure the turbulence model continuity at the interface point I .

Results on the impact of the Spalart-Allmaras modeling strategy on the accuracy of the model are given in section 4.5.1.2.

4.2 Numerical implementation of the wall model

This section focuses on the numerical implementation of the wall model in the framework of a cell-centered finite volume discretization.

4.2.1 Wall model discretization

As detailed in Section 2.2.1, the model substitutes the near-wall solution with a modeled evolution in the near-wall region, Ω_W , while shifting the boundary condition to the interface Γ_I . This interface demarcates the modeled region from the rest of the domain, where conventional RANS integration occurs. In a discrete setting, Ω_W is typically defined as a fixed number of cell layers adjacent to the wall, with the interface Γ_I located at the top of this region. Within a cell-centered framework, the flow state at the interface (point I) is unknown, necessitating interpolation between the states of the nearest cells. Alternatively, a sampling point S can be established above the interface in the RANS region to derive the local flow state necessary for model input. Thus, in the wall model formulation presented in Section 3.1, point I at the interface is replaced by sampling point S . Figure 2.6 illustrates the adaptation of this wall model strategy in a discrete cell-centered framework.

The boundary conditions for RANS integration along the R-WM I. are determined by modeling flow values at ghost cells, which reside within the Ω_W region but belong to the numerical stencil of the RANS region across the interface. The number of ghost cells required depends on the numerical scheme utilized; in this work, two layers of ghost cells are implemented. The ghost cell configuration is depicted in figures 2.7 and 2.8.

4.2.2 Wall model application

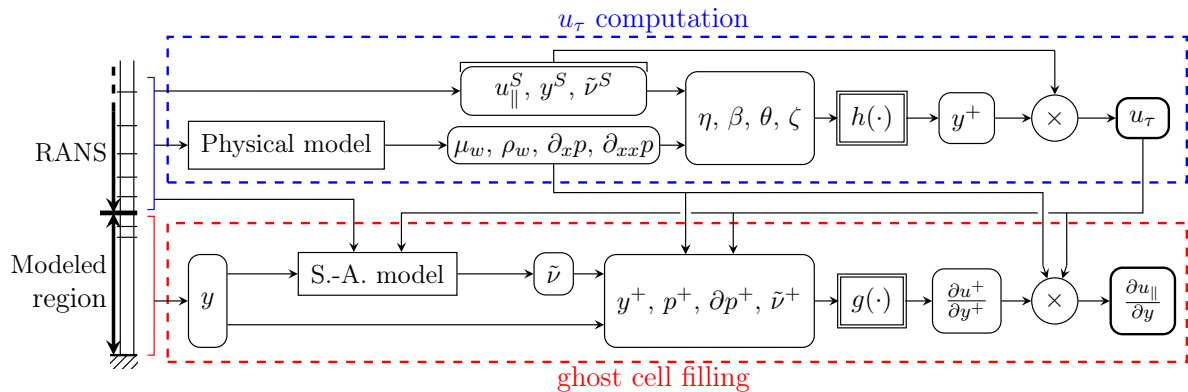


Figure 4.1: Schematic representation of wall model methodology to determine skin friction velocity and wall normal derivative of tangential velocity at the RANS-wall model interface.

The section explains the methodology adopted to apply the wall model and determine the evolution of tangential velocity u_{\parallel} in the near-wall region of the boundary layer. The cell index notation used is explained in figure 2.8a. The model fills the ghost cells belonging to the numerical stencil of the first computed cell in the resolved zone (j) to set the boundary condition at the RANS - Wall model interface, which is located at $j - \frac{1}{2}$.

The first step is to determine the thermodynamic state in the near-wall modeled region and the evolution of the pressure field along the streamwise direction. This is done as shown in section 4.1.2. Then, the model proceeds to the estimate of the local wall shear stress by

determining the skin friction velocity u_τ , which exploits the values at the sampling point, fixed at the first computed cell (j). The required set of data are the tangential velocity u_\parallel , the wall normal distance y and the Spalart-Allmaras variable $\tilde{\nu}$. These data as well as the pressure derivatives and the thermodynamic state are then combined to compute the dimensionless input features of the functional h , which yields the dimensionless wall distance y^+ at the sampling point j . This allows for the straightforward estimation of the skin friction velocity, given the wall distance of the sampling point.

Once the local shear stress has been determined, the boundary condition can be enforced by modeling the tangential velocity in the ghost cells, placed respectively at $j - 1$ and $j - 2$. In order to ensure the continuity of the modeled velocity profile, the model fixes the wall normal derivative of tangential velocity at the interface which can be expressed as

$$\left. \frac{\partial u_\parallel}{\partial y} \right|_{j-\frac{1}{2}} = \frac{u_{\parallel j} - u_{\parallel j-1}}{y_j - y_{j-1}}. \quad (4.18)$$

The estimation of this quantity is performed using the second functional g . The set of its input feature are obtained through the thermodynamic variables, the wall distance and the reconstructed Spalart-Allmaras variable, as explained in section 4.1.3. The value of the tangential velocity in the first ghost cell $j - 1$ is given by

$$u_{\parallel j-1} = u_{\parallel j} - \left. \frac{\partial u_\parallel}{\partial y} \right|_{j-\frac{1}{2}} (y_j - y_{j-1}), \quad (4.19)$$

which enforces the wall normal derivative of the tangential velocity at the interface. Then, the operation is repeated in order to fill the ghost cells located one step closer to the wall ($j - 2$). In this case, the functional g is used to estimate the derivative at $j - \frac{3}{2}$. In the configuration shown in figure 2.8b, the algorithm is repeated until the tangent velocity u_\parallel at the closest cell to the wall is determined.

Algorithm 2 summarizes the modeling procedure for the evolution of the tangential velocity profile u_\parallel .

4.2.3 Numerical effect of Dirichlet-To-Neumann approach

The proposed method replaces the Dirichlet boundary condition at the interface with a Neumann boundary condition, thereby fixing the derivative of the wall's tangential velocity. Some may argue that DtN formalism is unnecessary, as the Dirichlet method (i.e., imposing directly the tangential velocity) could still be utilized by just replacing the iterative estimation of the wall shear stress with a direct estimation applying the functional h . This reduces to

$$u_\tau = \frac{\nu_w}{y} h(\eta, \beta), \quad (4.20)$$

followed by the modeling of the wall tangent velocity profile

$$u_\parallel = u_\tau u^+ = \frac{\nu_w}{y} h(\eta, \beta) f(y^+, p^+). \quad (4.21)$$

as in Romanelli et al. [92] and showed in chapter 3. However, this method results in instabilities and convergence difficulties. Since, unlike the iterative estimation of wall shear stress, this approach does not guarantee the continuity of the velocity profile between the modeled and resolved zone.

Figure 4.2 illustrates a qualitative example in which the same function f is used to model the near-wall evolution of the tangential velocity u_\parallel for two different approaches to

Algorithm 2: Modeling of tangential velocity u_{\parallel} . Sampling point S placed at j cell.

Inputs : $u_{\parallel}^S, y^S, \tilde{\nu}^S, \frac{\partial p}{\partial x}, \frac{\partial^2 p}{\partial x^2}$

Estimate ρ_w, μ_w through physical model in section 4.1.2

Compute $\eta, \beta, \theta, \zeta \leftarrow \frac{u_{\parallel}^S y^S}{\nu_w}, \frac{y^{S3}}{\rho_w \nu_w^2} \frac{\partial p}{\partial x}, \frac{y^{S4}}{\rho_w \nu_w^2} \frac{\partial^2 p}{\partial x^2}, \frac{\tilde{\nu}^S}{\nu_w}$ (equation (4.10))

Estimate $y^{+S} \leftarrow h(\eta, \beta, \theta, \zeta)$

Compute $u_{\tau} \leftarrow \frac{\nu_w}{y^S} y^{+S}$ (equation 4.7)

if configuration in figure 2.8a **then**

 | $N = 2$ (2 ghost cells are modeled)

else

 | N equal to the number of cells below the RANS-wall model interface

end

for $n \leftarrow 1$ to N **do**

 | **Inputs :** $y_{j+\frac{1-2n}{2}}, y_{j+n}$

 | Compute $y_{j+\frac{1-2n}{2}}^+, p^+, \partial p^+ \leftarrow \frac{u_{\tau} y_{j+\frac{1-2n}{2}}}{\nu_w}, \frac{\nu_w}{\rho_w u_{\tau}^3} \frac{\partial p}{\partial x}, \frac{\nu_w^2}{\rho_w u_{\tau}^4} \frac{\partial^2 p}{\partial x^2}$ (equation 4.11)

 | Estimate $\tilde{\nu}_{j+\frac{1-2n}{2}}^+ \leftarrow \sum_{k=0}^3 a_k y_{j+\frac{1-2n}{2}}^{+k}$ (equation 4.16)

 | Estimate $\frac{\partial u^+}{\partial y^+} \Big|_{j+\frac{1-2n}{2}} \leftarrow f\left(y_{j+\frac{1-2n}{2}}^+, p^+, \partial p^+, \tilde{\nu}_{j+\frac{1-2n}{2}}^+\right)$

 | Compute $\frac{\partial u_{\parallel}}{\partial y} \Big|_{j+\frac{1-2n}{2}} \leftarrow \frac{u_{\tau}^2}{\nu_w} \frac{\partial u^+}{\partial y^+} \Big|_{j+\frac{1-2n}{2}}$

 | Integrate for $u_{\parallel j-n} \leftarrow u_{\parallel j-n+1} - \frac{\partial u_{\parallel}}{\partial y} \Big|_{j+\frac{1-2n}{2}} (y_j - y_{j-n})$ (equation 4.19)

end

estimate skin friction velocity. The iterative algorithm ensures that the resulting velocity profile is consistent with the data at the sampling point. This is achieved by iteratively adjusting the estimated skin friction velocity u_{τ} until the modeled velocity profile closely matches the observed data. This ensures near continuity at the interface between the modeled and resolved velocity profile. In contrast, the direct estimation through the application of the functional h does not observe any constraints on the modeled velocity profile at the sampling point S . As a consequence, the velocity profiles generated by direct estimation method may deviate from the observed data at the sampling point, potentially leading to a significant discrepancy between the modeled and resolved velocity profiles. This discrepancy can lead to numerical issues and instabilities. In such cases, a nonphysical shear stress is calculated across the interface, particularly during the initial stages of computation. This affects the viscous flux between the modeled and resolved zones, hindering the simulation's ability to converge.

The DtN approach, instead of the modeling strategy in equation 4.21, fixes a physical constraint on the viscous stress across the wall model interface, through a Neumann boundary condition by imposing the shear stress

$$\tau(y) = \mu \frac{\partial u_{\parallel}}{\partial y}(y). \quad (4.22)$$

which integrated for the wall tangent velocity allows to ensure continuity across the interface.

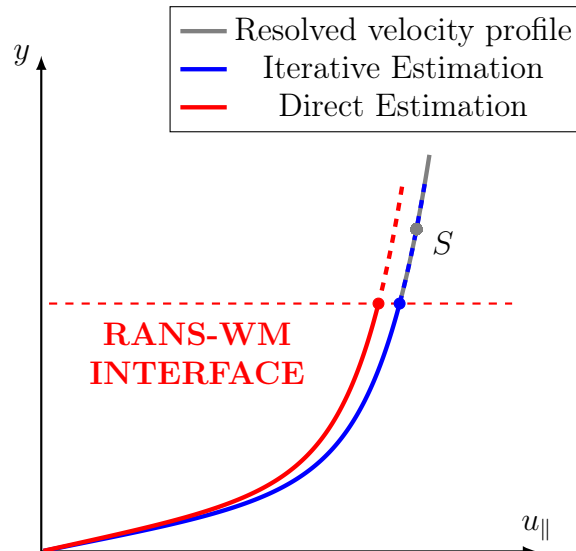


Figure 4.2: Scheme of wall tangential velocity u_{\parallel} modeling. Comparison of iterative estimation for skin friction velocity u_{τ} of the previous approach [92] and the direct approach through functional h .

4.3 Flow configurations

This section introduces the primary flow configurations used for training and evaluating the data-driven wall model. Reference and training data come from a series of fine wall-resolved RANS simulations. The model’s performance is assessed by comparing wall-modeled simulations with wall-resolved reference simulations.

The configurations used for this purpose are the 2D bump case and the airfoil case, both discussed in section 2.3.

4.3.1 Bump flow case

The flow configuration used for training and first evaluation of performances of the current model is the bi-dimensional bump case, as described in section 2.3.1.

4.3.1.1 Training and testing datasets

Various flow configurations are employed for training and testing the wall model, primarily generated by varying the bump height and the Reynolds number of the flow.

These flow configurations and definitions are similar to those from chapter 3, which allows for a close comparison of their results with the novel strategy proposed in this chapter. However, the present study considers a larger data set, including near-separation cases for higher Reynolds numbers, ranging from $Re = 10^6$ to $Re = 10^7$. The bump heights h considered vary from 0.02 to 0.08. Additionally, a degenerated bump ($h = 0$), representing a flat plate geometry, is included as a training case. Figure 4.3 shows the (Re, h) -combinations considered in this work.

A total of 50 (Re, h) -combinations are considered, with 23 allocated for training and 27 for evaluating model performance. The training configurations cover a broad range of boundary layer conditions, from zero-pressure gradient to near separation boundary layers, with an emphasis on near-stall scenarios and configurations close to the flow separation limit.

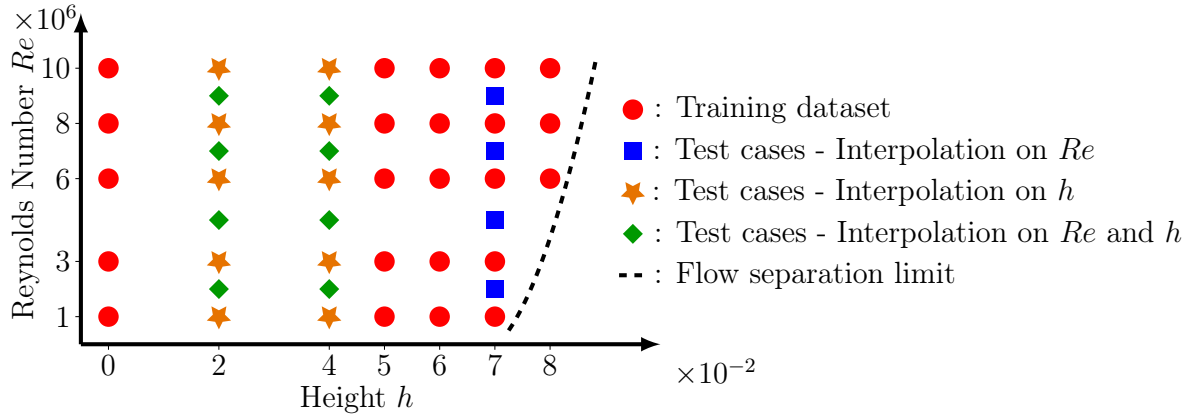


Figure 4.3: Training and test datasets obtained from different combinations of Reynolds number Re and bump height h .

As noted in Romanelli et al. [92], accuracy and convergence issues arise when neural network models operate beyond their training limits (a well-known limitation of most data-based approaches). Therefore, the test cases are restricted to interpolation conditions to ensure that the neural networks operate within the boundaries set by the training dataset in terms of the selected input features. Three distinct sets of evaluation cases are identified: (i) flow configurations with intermediate bump heights $h = 0.02$ and $h = 0.04$, matched with Reynolds numbers from the training dataset, (ii) A bump height of $h = 0.07$ and intermediate Reynolds numbers existing in the training dataset, (iii) configurations with both unseen bump heights and unseen Reynolds numbers.

The data extraction zone used for building the dataset is limited to $X \in [0.3, 1.2]$, as shown in figure 2.9b. Thus, the wall model performances are evaluated within this range.

4.3.2 Airfoil case

To evaluate the robustness of the proposed wall model, it is tested on a geometry entirely different from the one used during training, specifically the Wortmann airfoil FX60-100, previously introduced in section 2.3.2.

Similar to the bump case, different flow conditions are employed for testing. Each configuration is defined by a free-stream Mach number of $M = 0.2$ and a temperature of $T_\infty = 300K$. The varying Reynolds numbers Re , computed using the unitary length of the airfoil, are obtained by adjusting the free-stream density ρ_∞ . For the presented cases, Reynolds numbers of $Re = 6 \cdot 10^6$ and 10^7 are used, with the angle of attack fixed at $\alpha = 0^\circ$.

4.4 Data-driven modeling

This section focuses on the neural networks used in the model, their training procedure and the data treatment.

4.4.1 Neural networks

As shown in section 4.1.1.1, DtN approaches requires the knowledge of h and g . In this work, two neural networks are trained to model these functions, each of them receiving a

set of four dimensionless input features and outputting dimensionless quantities, as shown in equations (4.9), (4.10) and (4.11). A first neural network h is trained to predict the dimensionless wall distance y^+ . It allows to estimate the local wall shear stress through the skin friction velocity u_τ , as shown in equation (4.7). The second neural network g is trained to estimate the wall normal derivative of the dimensionless wall velocity u^+ as a function of the selected input features following classical wall shear stress scaling. An alternative approach is to train a neural network f to reproduce the evolution of the dimensionless velocity profile u^+ , as it was done in the previous approach from Romanelli et al. [92] in the chapter 3. The derivative with respect to the dimensionless wall distance y^+ can then be computed through algorithmic differentiation of the neural network. However, this strategy is not followed due to a lower accuracy of the model and higher computational cost. This point is further discussed in section 4.4.6.

Moreover, one may argue that a single function $F(\eta, \beta, \theta, \zeta)$ may be defined to describe the DtN map. From equation (4.8) and considering the additional parameters in section 4.1.1.1, it is possible to write

$$\frac{\partial u_{\parallel}}{\partial y} = \frac{\nu}{y^2} F(\eta, \beta, \theta, \zeta). \quad (4.23)$$

However, for the sake of interpretability, accuracy, and especially to ease the integration of the Spalart-Allmaras modeling in the near-wall region, this unified strategy is not followed. This choice is further discussed in section 4.4.7.

4.4.2 Neural network architecture

The model employs two feedforward neural networks (FNNs), each processing a set of input features and generating a single output. The neural network comprises an input layer, where the input features are fed; multiple hidden layers; and an output layer, which yields the estimated quantity. The number of nodes in the input and output layers is determined by the dimensionality of the input feature set and the number of quantities to be estimated, respectively. In this case, both neural networks, g and h , possess four nodes within the input layer and a single node within the output layer.

The architecture of the neural networks, including the number of nodes and the number of hidden layers, significantly affects the CPU cost of the CFD solver. Therefore, relatively small neural networks are employed. Both the g and h networks have architectures consisting of 4 hidden layers with 16 nodes each. Each node is fully connected to nodes in the preceding and succeeding layers. The weights and biases of these connections are determined through the learning process.

Furthermore, in consideration of the varying dynamic ranges of the input features a normalization layer is included between the input and hidden layers, as introduced in section 2.1.2.2.1. These normalization nodes are connected to their respective input features and are fully connected to the subsequent hidden layer. Prior to neural network training, the weights and biases of the normalization layer are determined to normalize each input feature (i.e., scaling to a range from 0 to 1) based on the training dataset. The figure 2.3 provides a schematic representation of the structure.

4.4.3 Dataset treatment

The training samples were drawn from the inner region of the boundary layer in the selected cases of flows over the bump geometry. These samples extended from the wall up

to $\frac{y}{\delta} \leq 0.15$, where δ is the boundary layer thickness. To facilitate the training process, the applicability range of the wall model was limited to a maximum $y^+ = 300$, which represents the highest dimensionless distance within the training samples.

Some of the input features of the g neural network present a highly non-linear behavior, especially in the near-separation case. This is evident in the cases of

$$p^+ \propto \frac{1}{u_\tau^3} \quad \text{and} \quad \partial p^+ \propto \frac{1}{u_\tau^4}, \quad (4.24)$$

where both features increase rapidly as the skin friction coefficient, C_f , and consequently the skin friction velocity u_τ approaches zero. This results in a highly uneven distribution of these input features across the dataset, with the majority of samples having near-zero values, while few of them presenting large values. To assist the training process, different scaling functions have been proposed, which are introduced in section 2.1.2.1.3.

For the power scaling in equation 2.2, the p exponent has been selected to be equal to $\frac{1}{3}$ and $\frac{1}{4}$ for p^+ and ∂p^+ , respectively. However, this scaling function, having the drawback of an infinite slope at zero, leads to a spaced distribution of samples in the near-zero region, which results in poor performance for near zero-pressure gradient applications. On the contrary, the Yeo-Johnson transformation [129], scaling the dataset in order to obtain a sample distribution close to a normal distribution, has the disadvantage of excessively clipping the evolution of the scaled input features for near-separation cases. This can cause poor performance of the model, when these conditions are met. Finally, the logarithmic scaling function in (2.4) is selected, setting the coefficient p to unity. This transformation function allows in fact to expand the range of values spanned for near-zero pressure gradient boundary layer samples, while keeping a satisfactory distribution of samples for the near separation samples.

An example of the application of the different scaling functions to the input feature p^+ is presented in figure 4.4. This figure shows the input feature and its scaled version plotted along the bump geometry for a flow configuration with $h = 0.08$ and $Re = 8 \cdot 10^6$. To ease comparison, all plotted values are normalized through min-max scaling.

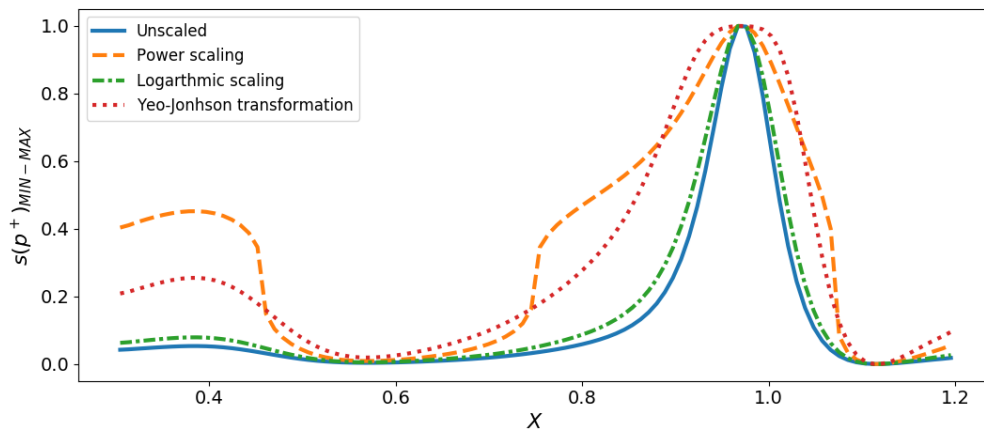


Figure 4.4: Comparison between the scaled and unscaled input feature p^+ for a flow over a bump geometry ($h = 0.08$ and $Re = 8 \cdot 10^6$).

The remaining input features of the neural network g as well as the input features of h are directly fed to the training process without needing a scaling technique.

4.4.4 Loss function selection

The neural networks are trained by minimizing a loss function, denoted as ϵ , which measures the discrepancy between the training data and the predictions made by the neural networks. This loss function is structured around the relative error, enabling equitable consideration of output values spanning various orders of magnitude. Specifically, for neural network g , the relative error, ϵ'_g , is calculated by comparing the dimensionless wall-normal derivative of the tangential velocity, $\frac{\partial u^+}{\partial y^+}$, with the corresponding reference values:

$$\epsilon'_g = \frac{\frac{\partial u^+}{\partial y^+} i - \frac{\partial u^+}{\partial y^+} i_{,\text{ref}}}{\frac{\partial u^+}{\partial y^+} i_{,\text{ref}}}. \quad (4.25)$$

Similarly, for neural network h , the relative error, ϵ'_h , is determined based on the predicted dimensionless wall distance, y^+ , relative to the reference values:

$$\epsilon'_h = \frac{y_i^+ - y_{i,\text{ref}}^+}{y_{i,\text{ref}}^+}. \quad (4.26)$$

The loss function is thus computed as

$$\epsilon = \frac{1}{N^S} \sum_{i=1}^{N^S} w_{\rho i} \mathcal{B}(\epsilon'), \quad (4.27)$$

where \mathcal{B} is the BerHu function [63] computed on the relative error and the coefficient $w_{\rho i}$ is a weighting scalar that accounts for the uneven sample distribution in the dataset. The BerHu function is defined as

$$\mathcal{B}(\epsilon) = \begin{cases} |\epsilon'| & |\epsilon'| \leq L \\ \frac{\epsilon'^2 + L^2}{2L} & |\epsilon'| > L \end{cases} \quad (4.28)$$

where the coefficient L is adopted equal to 1. This loss function is essentially the reverse form of the Huber loss [52]. The BerHu loss allows in fact to heavily punish relevant errors through its quadratic evolution, while maintaining sensitivity to small errors through a linear evolution for near-zero errors. Figure 4.5 depicts a qualitative representation of the BerHu loss function, which is compared to both a quadratic and a linear evolution of relative errors.

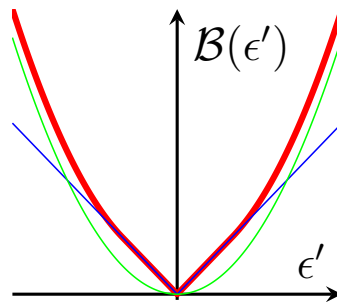


Figure 4.5: Qualitative representation of BerHu function (*red*) and comparison with linear (*blue*) and quadratic (*green*) evolution.

The weighting coefficient $w_{\rho i}$ is computed to address the uneven distribution of samples in the regression dataset [107]. A lower sample coefficient is computed for common samples

across the training dataset, while rarer ones get higher values of the weighting coefficient. It is computed as in equation (2.7), based on the min-max normalized local density of the training dataset. The latter is computed on a standardized training dataset (i.e. zero mean and unitary standard deviation), using a KDE technique through Gaussian kernels with unitary standard deviation as introduced in section 2.1.2.2.3. In figure 4.6, an overview of the obtained sample weight is shown respectively for the neural network g and h . Most common samples are respectively found for near zero-pressure gradient boundary layers present in the dataset.

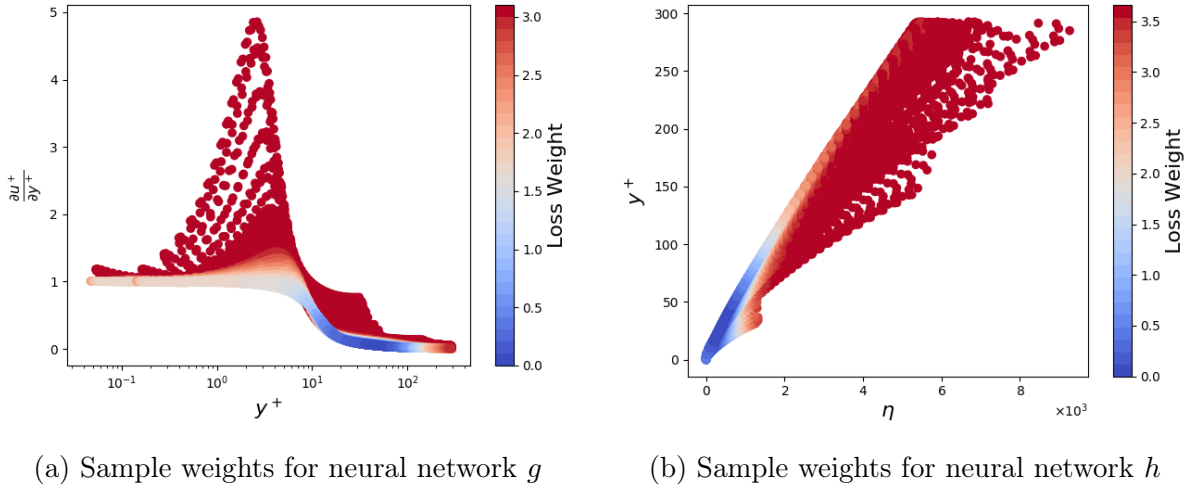


Figure 4.6: Overview of obtained sample weights used during neural network training to compute the loss function.

4.4.5 Training strategy

The neural networks are trained using the gradient-based Adam algorithm [59], implemented in the TensorFlow deep learning library [28]. The training process utilized 85% of the available data, reserving the remaining 15% for validation to monitor over-fitting. More details on the neural network training strategy are given in 2.1.2.2.4. The evolution of training and validation loss during the training process is given in figure 4.7. The trained neural networks are then selected at epochs 1507 and 3475 for h and g respectively.

4.4.6 Discussion on direct estimation and derivative computation

The proposed wall model uses a neural network g to estimate the wall-normal derivative of the dimensionless velocity $\partial u^+/\partial y^+$. There are two potential approaches for this estimation. The first, more straightforward approach, involves directly training the neural network to estimate $\partial u^+/\partial y^+$, which is the chosen method in this study. The second approach trains the neural network f to predict the dimensionless velocity u^+ itself, and then differentiates the output with respect to the wall-normal direction to be used in the model. This latter approach is more conventional, as the neural network focuses on estimating the dimensionless velocity rather than its derivative.

This section compares these two approaches to justify the chosen method. First, the training methodology for the alternative approach is outlined. Then, the accuracy at the end of training is assessed and compared to the selected approach. Finally, the computational cost of neural network inferences for both methods is evaluated.

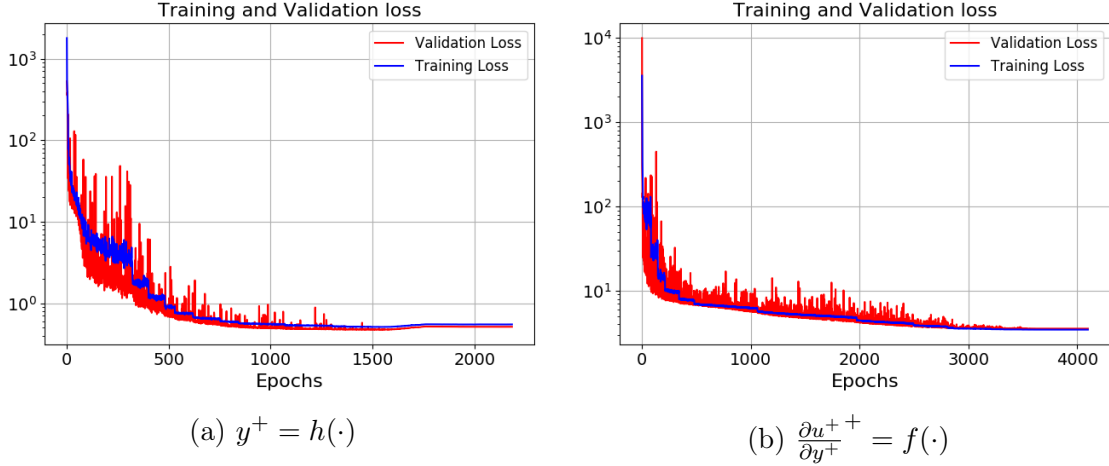


Figure 4.7: Training and validation loss evolution during training process of h and g neural networks.

For the approach where the neural network is trained to estimate the dimensionless velocity u^+ , the loss function minimized during the learning process is

$$\epsilon = \frac{1}{N^S} \sum_{i=1}^{N^S} w_{\rho i} \frac{1}{2} \left(\mathcal{B}(\epsilon_{u^+}) + \mathcal{B}(\epsilon_{\frac{\partial u^+}{\partial y^+}}) \right). \quad (4.29)$$

This loss function comprises the sum of BerHu functions applied to the absolute percentage error of both the dimensionless velocity u^+ and its wall-normal derivative. The relative errors are calculated as follows:

$$\epsilon_{u^+} = \frac{u_i^+ - u_{i,\text{ref}}^+}{u_{i,\text{ref}}^+} \quad \text{and} \quad \epsilon_{\frac{\partial u^+}{\partial y^+}} = \frac{\frac{\partial u^+}{\partial y^+}_i - \frac{\partial u^+}{\partial y^+}_{i,\text{ref}}}{\frac{\partial u^+}{\partial y^+}_{i,\text{ref}}}. \quad (4.30)$$

Both neural networks share the same architecture presented in the paper. The approaches are evaluated using the Mean Absolute Percentage Error (MAPE) and the Pearson correlation coefficient, computed on the dimensionless velocity u^+ and its wall-normal derivative, with reference to the training dataset. In the $\partial u^+/\partial y^+ = g(\cdot)$ approach, the dimensionless velocity is obtained by integrating the estimated wall-normal derivative starting from the zero value at the wall. In contrast, the $u^+ = f(\cdot)$ approach requires algorithmic differentiation of the neural network to estimate the value of the derivatives.

Table 4.1 presents the computed accuracy of both methods, while figure 4.8 shows the correlation of the obtained neural networks.

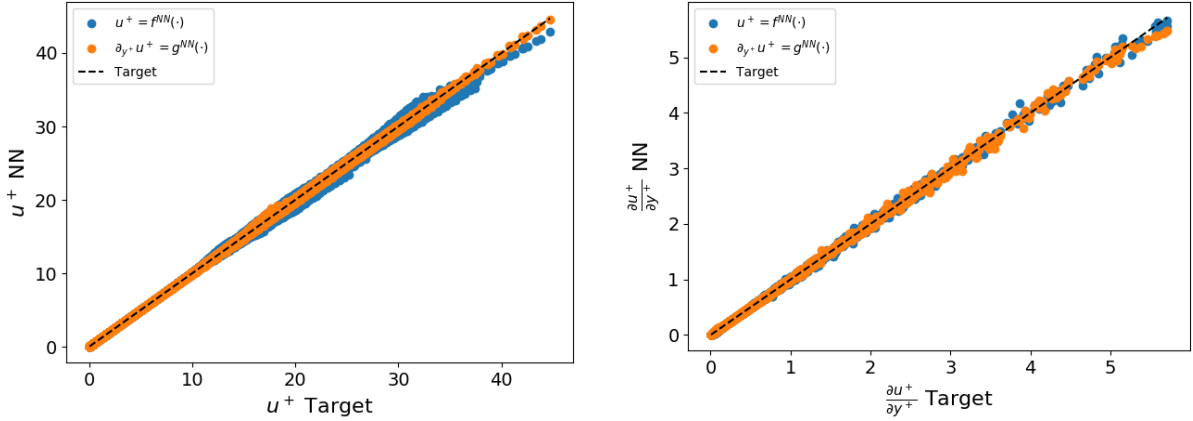
The $\partial u^+/\partial y^+ = g(\cdot)$ approach demonstrates greater accuracy for both the dimensionless velocity u^+ and its derivative compared to the alternative approach. Additionally, computing the wall-normal derivative of the dimensionless velocity u^+ is approximately 80% more computationally expensive using the $u^+ = f(\cdot)$ approach.

4.4.7 Discussion on single and multiple neural networks approach

The definition of the methodology for the fully data-driven wall model involves directly modeling the wall-normal derivative of the tangential velocity, $\partial u_{\parallel}/\partial y$, using either a

Test of $u^+ = f_{NN}$ and $\partial_{y^+}u^+ = g_{NN}$ on training dataset				
	$u^+ = f_{NN}$		$\partial_{y^+}u^+ = g_{NN}$	
	u^+	$\partial_{y^+}u^+$	u^+	$\partial_{y^+}u^+$
MAPE [%]	0.499	1.617	0.334	1.357
Correlation [%]	99.966	99.992	99.992	99.992

Table 4.1: Errors and correlation between neural networks output and training dataset. Comparison between the neural network $u^+ = f_{NN}$ and $\partial_{y^+}u^+ = g_{NN}$.



(a) Correlation between predicted dimensionless velocity u^+ with training dataset.

(b) Correlation between predicted wall normal derivative of dimensionless velocity $\frac{\partial u^+}{\partial y^+}$ with training dataset.

Figure 4.8: Correlation between predicted dimensionless velocity u^+ and wall normal derivative of dimensionless velocity $\frac{\partial u^+}{\partial y^+}$ with training dataset. Comparison between the neural network $u^+ = f_{NN}(\cdot)$ and $\partial_{y^+}u^+ = g_{NN}(\cdot)$.

single neural network or a modular approach with two neural networks, as shown in the following equation:

$$\frac{y^2}{\nu} \frac{\partial u_{\parallel}}{\partial y} = F(\eta, \beta, \theta, \zeta) \quad \text{or} \quad \frac{y^2}{\nu} \frac{\partial u_{\parallel}}{\partial y} = h^2(\eta, \beta, \theta, \zeta)g(p^+, y^+, \partial p^+, \tilde{\nu}^+), \quad (4.31)$$

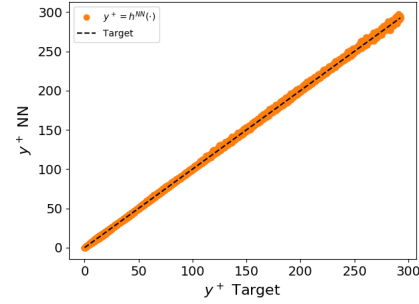
where the latter is the chosen option.

To justify the chosen strategy, the accuracy of the different approaches is evaluated by comparing the modular approach to the single neural network approach. A new neural network is thus trained to directly estimate the wall-normal derivative of the tangential velocity $\partial u_{\parallel}/\partial y$ as shown in equation 4.31. The F neural network uses the same input features as the h neural network. To ensure a fair accuracy comparison, the architecture of F is designed by summing the hidden layer sizes of the h and g neural networks, thereby maintaining comparable computational costs for both approaches. Consequently, the neural network F consists of eight hidden layers with 16 nodes each.

Before comparing the accuracy of the single and multiple neural networks approaches, the post-training accuracy of the h neural network is evaluated. Figure 4.9 displays the Mean Absolute Percentage Error (MAPE) and the Pearson correlation coefficient relative to the training dataset.

Training results of $y^+ = h_{NN}(\cdot)$	
MAPE [%]	0.458
Correlation [%]	99.998

(a) Errors and correlation of the predicted $y^+ = h_{NN}(\cdot)$ with reference to the training dataset



(b) Correlation between predicted dimensionless velocity $y^+ = h_{NN}(\cdot)$ with training dataset.

Figure 4.9: Errors and correlation of neural networks $y^+ = h_{NN}(\cdot)$ and training dataset.

The Mean Absolute Percentage Error (MAPE) and the Pearson correlation coefficient for both the single neural network and multiple neural networks approaches, with reference to the training dataset, are presented in table 4.2. Additionally, figure 4.10 illustrates the correlation for both approaches.

Test of training strategy $\frac{y^2 \partial_y u}{\nu} = F_{NN}(\cdot)$ and $\frac{y^2 \partial_y u}{\nu} = h_{NN}^2(\cdot)g_{NN}(\cdot)$ on training dataset		
	$F_{NN}(\cdot)$	$h_{NN}^2(\cdot)g_{NN}(\cdot)$
MAPE [%]	1.792	1.444
Correlation [%]	99.962	99.914

Table 4.2: Errors and correlation between neural networks output and training dataset. Comparison between the single neural network strategy, where $\frac{y^2 \partial_y u}{\nu} = F_{NN}(\cdot)$; and the modular neural network strategy where $\frac{y^2 \partial_y u}{\nu} = h_{NN}^2(\cdot)g_{NN}(\cdot)$.

The results indicate that both approaches exhibit very close accuracy performance. The modular approach, composed by the two neural networks h and g , is chosen to enhance the interpretability of the model and facilitate the integration of its different components. For the near-wall reproduction of the Spalart-Allmaras model, the estimation the local skin friction velocity u_τ is required. The single neural network approach does not directly provide this information, necessitating additional techniques to determine it when needed. Therefore, the modular approach is preferred for its practical benefits in model integration and interpretability.

4.5 Results

4.5.1 Results on bump geometry

4.5.1.1 Test procedure

This section presents the results obtained using the proposed wall model on the various flow configurations and geometries defined in section 4.3.1.1. These results are compared with the wall-resolved RANS simulation.

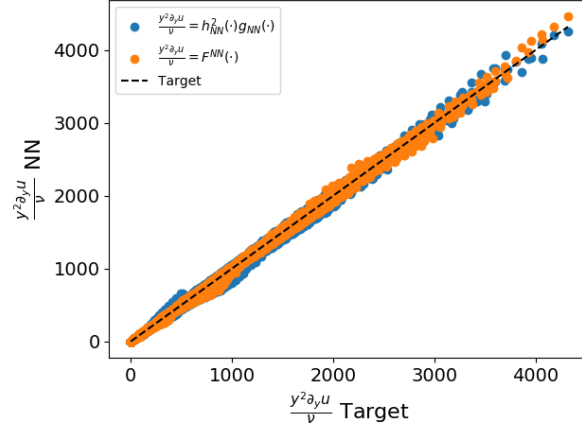


Figure 4.10: Correlation between predicted dimensionless wall normal velocity derivative $\frac{y^2 \partial_y u}{\nu}$ with training dataset. Comparison between the single neural network strategy, where $\frac{y^2 \partial_y u}{\nu} = F_{NN}(\cdot)$; and the modular neural network strategy where $\frac{y^2 \partial_y u}{\nu} = h_{NN}^2(\cdot)g_{NN}(\cdot)$.

The wall model is applied only to established turbulent boundary layers. Consequently, the wall-modeled computation is confined to the bump geometry area and downstream, specifically for $X \in [0.3, 1.5]$. The upstream section of the wall, i.e. $X \in [0, 0.3]$, is fully resolved. The wall model is applied on a structured grid as detailed in section 4.2. The numerical stencil of the spatial scheme in the solver includes two cell layers below the RANS interface, set at a constant distance from the wall surface. Thus, two layers of cells below the interface are modeled; lower cells along the wall-normal direction are unaffected by the modeling approach during the RANS computation. For $X = 0.3$, the configuration in 2.8b is applicable, where the fully resolved section meets the modeled one. Here, the modeled cells reach the wall, in order to act as ghost-cell in the streamwise direction for the fully resolved area. The sampling point is set at the first RANS-computed cell above the interface.

For each simulation, a target maximum y_{MAX}^+ is set for the interface. If the target y_{MAX}^+ exceeds the inner region of the boundary layer (i.e., $\frac{y}{\delta} \leq 0.15$), the interface is adjusted to meet this condition, resulting in a lower effective value of y^+ . Table 4.3 shows the cases where the interface is limited by the condition $\frac{y}{\delta} \leq 0.15$.

	y_{MAX}^+ at RANS interface for configuration limited at $\frac{y}{\delta_{MAX}} \approx 0.15$						
	$h = 0$	$h = 0.02$	$h = 0.04$	$h = 0.05$	$h = 0.06$	$h = 0.07$	$h = 0.08$
$Re = 10^6$	50	60	60	70	70	80	
$Re = 2 \cdot 10^6$		90	110			130	
$Re = 3 \cdot 10^6$	110	130	150	160	170	180	

Table 4.3: Values of the maximum dimensionless wall distance y_{MAX}^+ encountered at the RANS-wall model interface for all the configuration of flows over the bump which are limited by $y/\delta \leq 0.15$ condition.

In order to ease the convergence of the computation and avoid neural network extrapolation, the simulations are initialized with a constant flow field and 10'000 solver iterations are performed using the Spalart-Allmaras wall law [110] to model the wall tangential velocity $u_{||}$ before switching to the data-based models. More information about the Spalart-Allmaras wall law are given in section 1.4.1.2.1.

Wall model performance is evaluated on the bump geometry only, from $X = 0.3$ to $X = 1.2$, which corresponds to the training data extraction zone.

First, the impact of the modeling strategy for the near-wall behavior of the Spalart-Allmaras model is assessed. Then, the following subsections present the model results on the training configurations as validation cases and a comparison with the iterative approach. Subsequently, the interpolation test cases are addressed for both Reynolds number and bump height interpolation.

A performance evaluation is conducted using both the 2-norm global error and the local normalized error on the skin friction coefficient C_f when compared to the fully-resolved RANS simulation. The global 2-norm error e_2 is obtained as follows

$$e_2 = \frac{\|C_f(X_i) - C_{f,ref}(X_i)\|_2}{\|C_{f,ref}(X_i)\|_2} = \frac{\sqrt{\sum_i [C_f(X_i) - C_{f,ref}(X_i)]^2}}{\sqrt{\sum_i [C_{f,ref}(X_i)]^2}}, \quad (4.32)$$

where X_i refers to the equally spaced solution points in the X -coordinate direction along the portion of the wall under consideration, C_f is the friction coefficient estimation derived from the wall model, while $C_{f,ref}$ is the friction coefficient estimation derived from the wall-resolved simulation.

Meanwhile, the local normalized error is computed as

$$e(X_i) = \frac{C_f(X_i) - C_{f,ref}(X_i)}{\overline{C_{f,ref}}}, \quad (4.33)$$

in which $\overline{C_{f,ref}}$ is the mean value of the friction coefficient estimation from the wall-resolved simulation along the evaluated area of the wall, i.e. $X \in [0.3, 1.2]$. The local value is not used for normalization purposes, as this would result in an artificial divergence of the error when C_f is close to zero.

4.5.1.2 Assessment of Spalart-Allmaras modeling strategy

In order to evaluate the impact of the modeling strategy for the near-wall behavior for the Spalart-Allmaras turbulence model, the proposed wall law is tested using both the linear evolution and the polynomial regression for the dimensionless Spalart-Allmaras variable $\tilde{\nu}^+$ as described in section 4.1.3.

The test is performed on the bump case with configuration $h = 0.06$ and $Re = 6 \cdot 10^6$. The RANS - Wall model interface is set at different distances from the wall, resulting in a maximum dimensionless wall distance of respectively $y_{MAX}^+ \approx 10, 50, 100, 150, 200$ along the bump geometry.

Figure 4.11 presents a comparison between the linear evolution and polynomial regression methods for the dimensionless Spalart-Allmaras variable, compared with a reference solution obtained from a fully resolved RANS simulation. The figure depicts the near-wall evolution of the dimensionless Spalart-Allmaras model for these approaches. The behavior of the Spalart-Allmaras variable is shown at $X \approx 0.9$. At this point, the RANS - Wall model interface is set locally at $y^+ \approx 4, 14, 23, 47, 98$. As anticipated, the linear evolution model exhibits accuracy only near the wall, rapidly losing accuracy as the wall distance increases. Furthermore, an abrupt discontinuity is evident at the interface between the modeled and computed Spalart-Allmaras variable. In contrast, the polynomial regression approach demonstrates a significantly better behavior. It not only accommodates higher modeling distances more effectively, but also resolves the discontinuity issue between the model and the RANS computation.

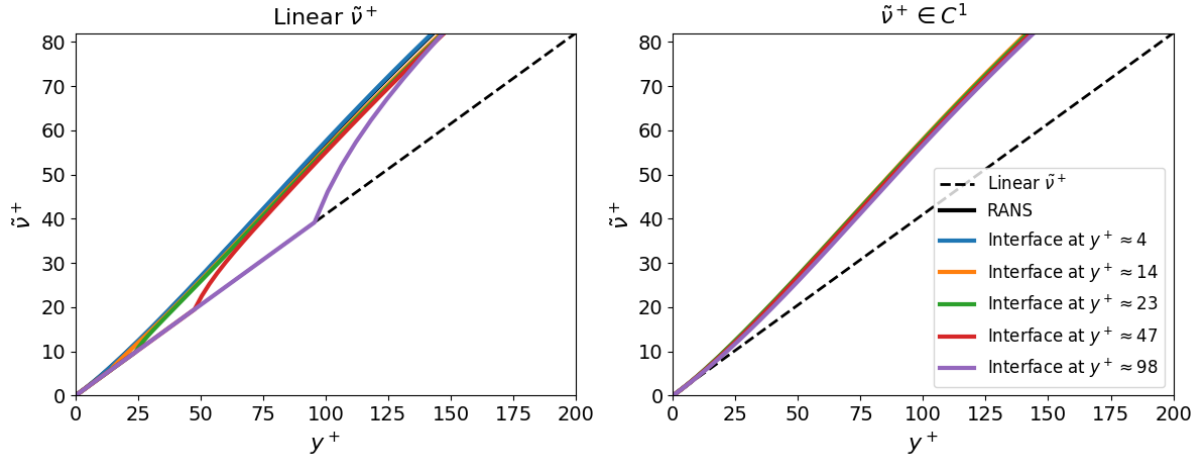


Figure 4.11: Comparison of the dimensionless Spalart-Allmaras variable \tilde{v}^+ for the classical linear model (*left*) and the polynomial regression approach (*right*). Bump case ($h = 0.06$ and $Re = 6 \cdot 10^6$) at $X \approx 0.9$. Maximum dimensionless distance of the first RANS computed cell: $y_{MAX}^+ \approx 10, 50, 100, 150, 200$. Locally, the RANS-Wall model interface is set at $y^+ \approx 4, 14, 23, 47, 98$.

The evaluation also includes the estimation of the friction coefficient C_f in comparison to the fully-resolved RANS simulation, which is achieved through the computation of a 2-norm error along the entire length of the bump surface (i.e., X coordinate ranging from 0.3 to 1.2 of the simulation domain). The results obtained are shown in figure 4.12. The table presents the values of the 2-norm global error on C_f , for different maximum dimensionless wall distances y_{MAX}^+ of the first RANS resolved cell, obtained using both the Spalart-Allmaras variable modeling approach. In the figure, the evolution of the e_2 error is plotted for the different modeling distances. The polynomial regression approach shows a near constant behavior across all tested modeling distances. On the contrary, the linear evolution model, despite exhibiting similar accuracy than the polynomial evolution model at low modeling distances (i.e., $y^+ \approx 10$), loses its accuracy at higher distances.

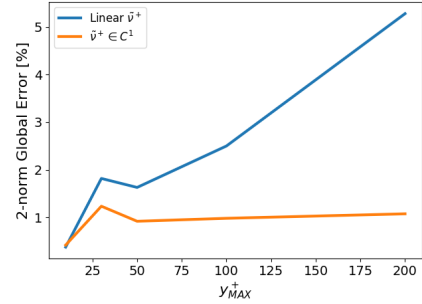
Given the previous results, the polynomial regression model is used in the following to evaluate the performances of the model using the fully data-driven approximation of the near-wall velocity evolution.

4.5.1.3 Discussion on the additional parameters for DtN-map

The data from the bump configuration reveal that considering only two input parameters (y^+, p^+) is actually not enough. This is seen along the downward slope of the bump geometry, where the dimensionless pressure gradient reaches its peak due to significant adverse pressure gradients. In this region, a certain number of stations along the bump geometry have the same local dimensionless pressure gradient (p^+), respectively upstream and downstream of the maximum value of the dimensionless pressure gradient. In these points, the boundary layer evolution presents slight differences. This is illustrated in figure 4.13, which depicts the variations in the evolution of the dimensionless tangential velocity, u^+ , and its wall-normal derivative $\frac{\partial u^+}{\partial y^+}$ at two locations downstream of the top of the bump. These locations exhibit identical adverse pressure gradient values (p^+), but opposite signs of ∂p^+ . Extra variables are thus needed. The second derivative of the pressure field allows the model to identify differences in boundary layer evolution, even when the dimensionless pressure gradients (p^+) are the same but their second derivatives differ.

Relative 2-norm global error on C_f .		
	Linear $\tilde{\nu}^+$	$\tilde{\nu}^+ \in C^1$
$y_{MAX}^+ \approx 10$	0.38%	0.42%
$y_{MAX}^+ \approx 30$	1.82%	1.24%
$y_{MAX}^+ \approx 50$	1.63%	0.92%
$y_{MAX}^+ \approx 100$	2.5%	0.98%
$y_{MAX}^+ \approx 200$	5.28%	1.08%

(a) Relative 2-norm global error on the skin friction coefficient C_f .



(b) Evolution of relative 2-norm global error on the skin friction coefficient C_f based on dimensionless wall distance of the modeled region.

Figure 4.12: Relative 2-norm global error on the skin friction coefficient C_f of the wall-modeled simulation with respect to reference wall-resolved RANS simulation and its evolution based on the dimensionless wall distance of RANS-Wall model interface. Comparison of classical linear model and the polynomial regression approach for the modeling of the dimensionless Spalart-Allmaras variable $\tilde{\nu}^+$. Bump case ($h = 0.06$ and $Re = 6 \cdot 10^6$). Maximum dimensionless distance of the first RANS computed cell: $y_{MAX}^+ \approx 10, 50, 100, 150, 200$.

Similarly, the boundary layer evolution varies depending on the local Reynolds number. The dimensionless Spalart-Allmaras variable $\tilde{\nu}^+$ is useful for correctly identifying these differences. An example is given in figure 4.14, which illustrates the dimensionless wall velocity u^+ and its wall normal derivative within a zero pressure gradient at two distinct locations characterized by different local Reynolds numbers. These locations exhibit slight variations in the velocity profiles, as well as in the behavior of the dimensionless Spalart-Allmaras variable.

As a result, the proposed wall model incorporates the two additional parameters mentioned above to characterize the state of the internal boundary layer. These parameters are thus based on the streamwise second derivative of the pressure, $\frac{\partial^2 p}{\partial x^2}$, and the Spalart-Allmaras variable, $\tilde{\nu}$. They are non dimensioned and integrated in the DtN-map operator as shown in section 4.1.1.1.

4.5.1.4 Model validation and comparison with iterative approach

The validation is carried out by comparing the results obtained with the model over flow configurations that belong to the training dataset to define a baseline for the model capabilities. Table 4.4 shows the 2-norm global error for the training dataset configurations, shown in figure 4.3. For the sake of conciseness, the table only displays the results obtained with the interface at the extreme values $y_{MAX}^+ \approx 50$ and 200, but the validation was also performed for $y_{MAX}^+ \approx 100$ (not shown), yielding similar results and exhibiting consistent behavior. In cases where the target y^+ condition was not reachable due to the limitation of the inner region of the boundary layer (i.e., $\frac{y}{\delta} \leq 0.15$), the RANS - Wall model interface was positioned closer to the wall. The actual achieved y_{MAX}^+ on the bump geometry is given in table 4.3.

The table illustrates the global consistency of the model throughout the training dataset, thereby validating the methodology and its training procedure. The errors range

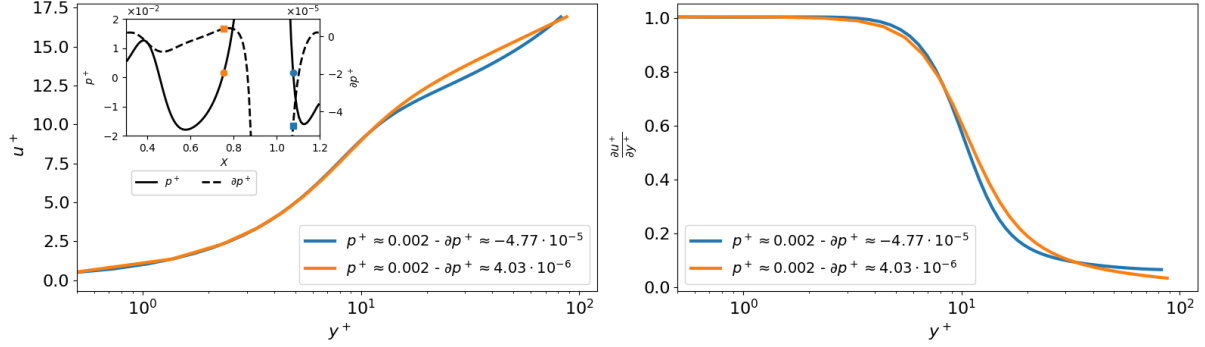


Figure 4.13: Evolution of dimensionless wall velocity u^+ and its derivative $\frac{\partial u^+}{\partial y^+}$ for boundary layer locations showing the same dimensionless pressure gradient p^+ and different values of dimensionless derivative of pressure gradient ∂p^+ .

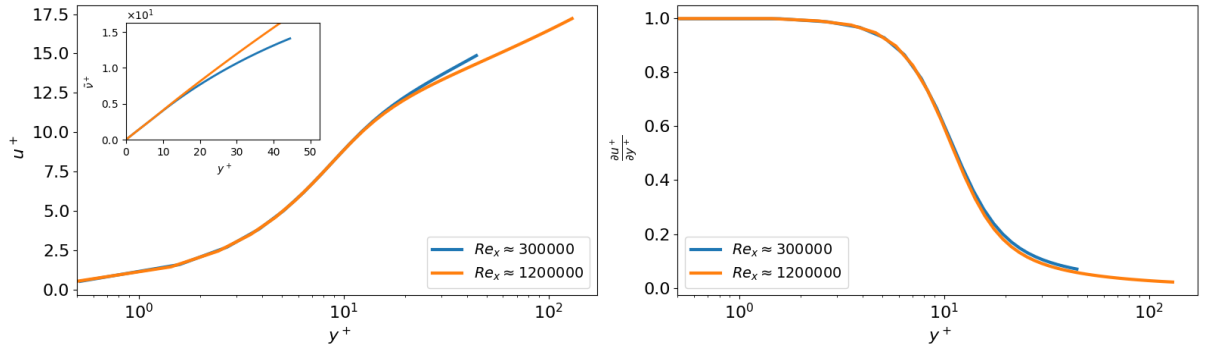


Figure 4.14: Evolution of dimensionless wall velocity u^+ and its derivative $\frac{\partial u^+}{\partial y^+}$ for zero-pressure gradient boundary layer for different local Reynolds numbers and \tilde{v}^+ profiles.

from 0.6% to 2.89%. Despite the observed consistency in the errors across varying bump heights, which affect the pressure gradient along the bump, a marginal increase in error is observed for the lowest Reynolds number configurations ($Re = 10^6$). This error increase may be attributed to the larger range of dimensionless pressure gradient, p^+ , and its relative derivative, ∂p^+ , encountered along the bump geometry.

The proposed model is compared to the iterative approach from [92]. To facilitate this comparison, the neural network within the model was retrained using the same dataset as the fully data-driven approach, following the methodology outlined in section 4.4. The model was then applied to the bump simulation with parameters $h = 0.06$ and $Re = 6 \cdot 10^6$, following the same strategy introduced in the previous section. The RANS - Wall model interface is placed at $y_{MAX}^+ \approx 10, 50$ and 200 .

Figure 4.15 shows the local error in the skin friction coefficient, C_f , with respect to the RANS resolved case, obtained through both the fully data-driven approach and the iterative approach for the three distinct modeling distances.

The fully data-driven approach appears to be largely unaffected by the increase in modeling distance, whereas the iterative approach exhibits a significantly different behavior depending on the wall distance. For $y_{MAX}^+ \approx 10$, both models demonstrate good accuracy, with a local error not exceeding 3%. However, our novel approach maintains far more consistent accuracy compared to the iterative approach, which shows oscillating accuracy along the streamwise direction. At other wall distances, the present strategy outperforms the iterative approach. The former maintains errors below 3%, whereas the latter errors rise

RANS interface at $y_{MAX}^+ \approx 50$					
	$h = 0$	$h = 0.05$	$h = 0.06$	$h = 0.07$	$h = 0.08$
$Re = 10^6$	2.84%	2.17%	2.5%	2.89%	
$Re = 3 \cdot 10^6$	0.7%	0.91%	0.95%	0.97%	
$Re = 6 \cdot 10^6$	0.67%	0.89%	0.92%	0.9%	0.97%
$Re = 8 \cdot 10^6$	0.59%	1.16%	1.18%	1.16%	1.39%
$Re = 10^7$	0.84%	0.77%	0.77%	0.73%	0.7%
RANS interface at $y_{MAX}^+ \approx 200$					
	$h = 0$	$h = 0.05$	$h = 0.06$	$h = 0.07$	$h = 0.08$
$Re = 10^6$	2.84%*	2.71%*	2.08%*	2.35%*	
$Re = 3 \cdot 10^6$	0.6%*	1.61%*	1.42%*	1.39%*	
$Re = 6 \cdot 10^6$	0.84%	0.94%	1.08%	1.17%	1.2%
$Re = 8 \cdot 10^6$	0.87%	1.35%	1.37%	1.41%	1.48%
$Re = 10^7$	1.42%	1.21%	1.11%	1.06%	1.04%

Table 4.4: 2-norm global error on the skin friction coefficient C_f between wall-modeled simulations and reference wall-resolved RANS for two different RANS-Wall model interface positions and different combinations of bump height and Reynolds number Re . Training cases on figure 4.3. The simulations where the interface is located for y_{MAX}^+ lower than target value to satisfy $y/\delta \leq 0.15$ are marked by *. The actual interface height is shown in table 4.3.

to approximately 10% and 14% for $y_{MAX}^+ \approx 50$ and 200, respectively. This improvement is primarily due to the treatment of the near-wall behavior of the Spalart-Allmaras model, as well as the additional input features considered in the approach presented in this chapter.

Table 4.5 presents the 2-norm global error for both models across varying modeling distances for the same case. The accuracy comparison aligns with previous findings. At the lowest modeling distance, both models exhibit similar performance. However, as the modeling distance increases, the iterative approach becomes less accurate, while our novel approach remains only slightly affected by the changes in modeling distance. The new strategy is therefore, on average, three times more accurate than the previous iterative approach.

Relative 2-norm global error on C_f .		
	Iterative approach	D2N-like
$y_{MAX}^+ \approx 10$	0.55%	0.42%
$y_{MAX}^+ \approx 30$	4.14%	1.24%
$y_{MAX}^+ \approx 50$	3.67%	0.92%
$y_{MAX}^+ \approx 100$	3.34%	0.98%
$y_{MAX}^+ \approx 200$	5.2%	1.08%

Table 4.5: Relative 2-norm global error on the skin friction coefficient C_f of the wall-modeled simulation with respect to reference wall-resolved RANS simulation. Bump case ($h = 0.06$ and $Re = 6 \cdot 10^6$). RANS-wall model interface placed at target $y_{MAX}^+ \approx 10, 30, 50, 100, 200$. Comparison between iterative approach and the new fully data-driven approach.

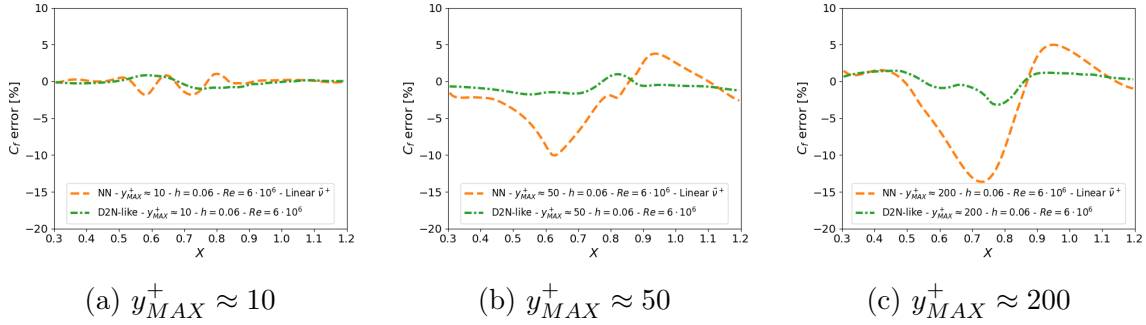


Figure 4.15: Relative local error on skin friction coefficient C_f between the reference wall resolved simulation and wall modeled ones. Bump case ($h = 0.06$ and $Re = 6 \cdot 10^6$). RANS-wall model interface placed at target $y_{MAX}^+ \approx 10, 50, 200$. Comparison between iterative approach and the new fully data-driven approach.

4.5.1.5 Interpolation in Reynolds number

This section examines the model interpolation capability. The performance is evaluated by testing the model on configurations not included in the training dataset (interpolation condition in Reynolds numbers). For this purpose, wall-modeled simulations are conducted with a bump height of $h = 0.07$ and varying Reynolds numbers Re . The Reynolds numbers considered are $Re = 2 \cdot 10^6$, $4.5 \cdot 10^6$, $7 \cdot 10^6$, and $9 \cdot 10^6$, representing intermediate values between those present in the training dataset.

Table 4.6 presents the 2-norm global error obtained by applying the model under interpolation conditions for the Reynolds number. The results are presented for a RANS interface at $y^+ \approx 50$ and 200 , with the intermediate modeling height of 100 excluded for the sake of brevity, given that no significant differences were observed for that height. For $Re = 2 \cdot 10^6$, the modeling height is limited to achieve $y_{MAX}^+ \approx 130$ to confine modeling to the inner region of the boundary layer.

Overall, the computed errors are low, ranging from 0.93% to 1.6% , and are only slightly affected by increasing the modeling height. These global error levels are comparable to those observed in the training dataset, thereby validating the model's generalization capabilities in interpolation conditions across different Reynolds numbers.

Figure 4.16 illustrates the skin friction coefficient, C_f , and its normalized local error, as defined in equation (4.33), for two interpolation cases: $h = 0.07$ with $Re = 2 \cdot 10^6$ and $Re = 9 \cdot 10^6$. These represent the lowest and highest Reynolds numbers tested. In both configurations, the model accurately predicts the skin friction coefficient, matching the RANS-resolved values along the bump geometry. The local error remains around $2\text{-}3\%$ along the geometry, with a peak at $X \approx 0.75$ (the top of the bump), reaching 7% for $Re = 2 \cdot 10^6$ and 5% for $Re = 9 \cdot 10^6$.

4.5.1.6 Interpolation in bump height

After assessing the interpolation capabilities for unseen Reynolds number, the model is tested in interpolation conditions over different bump heights. The model is thus tested on bump flow conditions with a Reynolds numbers existing in the training dataset, such as $Re = 10^6$, $3 \cdot 10^6$, $6 \cdot 10^6$, $8 \cdot 10^6$ and 10^7 and two distinct bump heights, $h = 0.02$ and $h = 0.04$, which were unseen during the training process.

Table 4.7 presents the 2-norm global error obtained for these configurations. As in the previous case, the results are shown for a RANS interface at $y^+ \approx 50$ and 200 , excluding

RANS interface at $y_{MAX}^+ \approx 50$	
$h = 0.07$	
$Re = 2 \cdot 10^6$	0.93%
$Re = 4.5 \cdot 10^6$	1.01%
$Re = 7 \cdot 10^6$	1.04%
$Re = 9 \cdot 10^6$	1.26%
RANS interface at $y_{MAX}^+ \approx 200$	
$h = 0.07$	
$Re = 2 \cdot 10^6$	1.62%*
$Re = 4.5 \cdot 10^6$	1.44%
$Re = 7 \cdot 10^6$	1.25%
$Re = 9 \cdot 10^6$	1.6%

Table 4.6: 2-norm global error on the skin friction coefficient C_f between wall-modeled simulations and reference wall-resolved RANS for two different RANS-Wall model interface positions and different combinations of bump height and Reynolds number Re . Test case simulations representing interpolation conditions on Reynolds number ($Re = 2 \cdot 10^6$, $4.5 \cdot 10^6$, $7 \cdot 10^6$ and $9 \cdot 10^6$). The simulations where the interface is located for y_{MAX}^+ lower than target value to satisfy $y/\delta \leq 0.15$ are marked by *. The actual interface height is shown in table 4.3.

the intermediate modeling height 100 for the sake of brevity. The cases presenting the two lowest Reynolds number configurations ($Re = 10^6$ and $3 \cdot 10^6$) required to limit the RANS interface to restrict the modeling to the inner region of the boundary layer. In the case of a bump presenting a height of $h = 0.02$, the maximum dimensionless wall distance available is set to $y_{MAX}^+ \approx 60$ and 150 respectively, while a bump height of $h = 0.04$ allows a modeling distance $y_{MAX}^+ \approx 60$ and 130 respectively. The computed 2-norm error values are consistent with those of the validation test, ranging between 0.47% and 2.71%. As for the validation tests, the errors for the lower Reynolds are found to be quite more relevant, since in these configurations larger values of dimensionless pressure gradient, p^+ , (adverse pressure gradient) and its relative derivative, ∂p^+ , are encountered.

Figure 4.17 illustrates the skin friction coefficient C_f and its normalized local error, as defined in equation (4.33), for the interpolation cases along the unseen bump heights: $h = 0.02$ and $h = 0.04$. The configurations are characterized by Reynolds numbers seen during training, ranging from $Re = 10^6$ to 10^7 . Two cases are shown: $h = 0.02$ at $Re = 10^6$ and $h = 0.04$ at $Re = 10^7$. These two cases present very different behaviors. The first one presents the highest error among the test dataset, indicating the challenging conditions encountered at the lowest considered Reynolds number. But one may notice that the accuracy remains high nonetheless. In this case, the local error reaches only 4% at most, but almost never descends below 2%. In contrast, the latter case demonstrates a remarkably better estimation of the skin friction coefficient. Even though the local error also peaks at 4% at the top of the bump ($X \approx 0.75$), it goes well below 2% for a significant portion of the bump geometry, thus explaining the significant difference in global error.

4.5.1.7 Interpolation in bump height and Reynolds number

Finally, this section tests the model accuracy in interpolation condition, both in Reynolds number and bump height. The model is tested for combinations of unseen bump height ($h = 0.02$ and $h = 0.04$) and Reynolds number ($Re = 2 \cdot 10^6$, $4.5 \cdot 10^6$, $7 \cdot 10^6$, and $9 \cdot 10^6$).

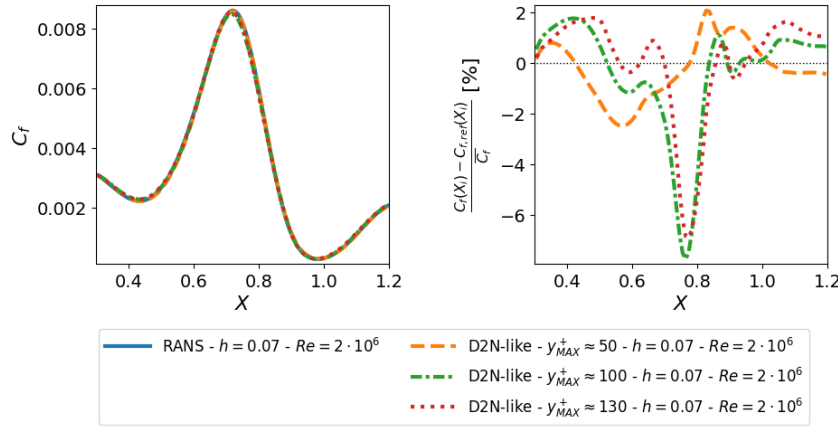
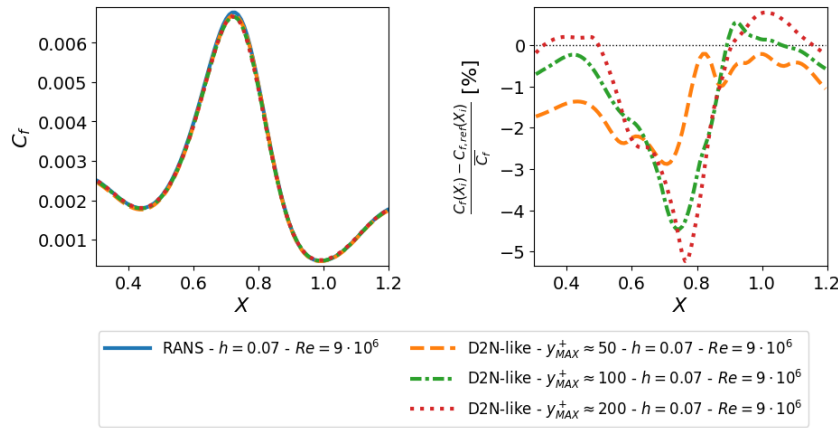
(a) Bump case $h = 0.04$ and $Re = 2 \cdot 10^7$ (b) Bump case $h = 0.07$ and $Re = 9 \cdot 10^6$

Figure 4.16: Skin friction coefficient C_f (left) and the relative local error (right) between the reference wall resolved simulation and wall modeled ones. Selected sample cases illustrating interpolation conditions across various Reynolds number with reference to training dataset. Fixed bump height $h = 0.07$. RANS-wall model interface placed at target $y_{MAX}^+ \approx 50, 100, 200$. When the target interface height does not respect $y/\delta < 0.15$, the interface is fixed at $y/\delta_{MAX} \approx 0.15$ and the obtained y_{MAX}^+ is shown in the legend.

Table 4.8 presents the 2-norm global error obtained by applying the fully data-driven model to the eight combinations considered. The computations were performed with the RANS interface at $y_{MAX}^+ \approx 50, 100$ and 200 . The results align with previous findings, with errors ranging from 0.54% to 1.76% and exhibiting a remarkably low dependency on the modeling distance.

Figure 4.18 illustrates the skin friction coefficient C_f and its normalized local error for two cases in interpolation condition in bump height and Reynolds number. The model shows good accuracy by closely reproducing the reference friction coefficient for both cases. The local error reaches a maximum of 3%.

4.5.2 Test on the airfoil geometry

This section presents the results of the proposed model applied to the Wortmann airfoil case. It starts by motivating the choice of the test case and the modeled region, and then presents the results.

RANS interface at $y_{MAX}^+ \approx 50$		
	$h = 0.02$	$h = 0.04$
$Re = 10^6$	2.44%	2.01%
$Re = 3 \cdot 10^6$	0.87%	0.86%
$Re = 6 \cdot 10^6$	0.47%	0.83%
$Re = 8 \cdot 10^6$	0.73%	1.08%
$Re = 10^7$	1%	1.32%
RANS interface at $y_{MAX}^+ \approx 200$		
	$h = 0.02$	$h = 0.04$
$Re = 10^6$	2.71%*	2.05%*
$Re = 3 \cdot 10^6$	1.25%*	1.61%*
$Re = 6 \cdot 10^6$	0.99%	0.92%
$Re = 8 \cdot 10^6$	1.43%	1.35%
$Re = 10^7$	2.14%	1.98%

Table 4.7: 2-norm global error on the skin friction coefficient C_f between wall-modeled simulations and reference wall-resolved RANS for two different RANS-Wall model interface positions and different combinations of bump height and Reynolds number Re . Test case simulations representing interpolation condition on bump height ($h = 0.02$ and $h = 0.04$). The simulations where the interface is located for y_{MAX}^+ lower than target value to satisfy $y/\delta \leq 0.15$ are marked by *. The actual interface height is shown in table 4.3.

4.5.2.1 Test setup and procedure

The Wortmann airfoil FX60-100 geometry is considered to evaluate the robustness of the proposed wall model. A particular attention was required for the airfoil leading edge, where the development of the boundary layer is at an early stage under conditions of pressure gradients at low local Reynolds numbers Re_x . Indeed, our data-based model was not trained for such conditions (the model is designed for well established turbulent boundary layers at a relatively high Reynolds number). Consequently, the model must be applied at a sufficient distance downstream from the leading edge, as shown in figure 2.11b. This point was considered in the airfoil selection process, as employing airfoils with large leading edge radii, such as NACA airfoils, would require to apply the model quite far downstream. The selected airfoil has a relatively small leading-edge radii, which helps to minimize significant pressure gradients near the leading edge.

The results obtained by the application of the model are compared to reference RANS simulation and to the analytical wall model by Spalart-Allmaras [110]. It is chosen as a reference because, by definition, it provides the most accurate approximation of the RANS-resolved boundary layer using the Spalart-Allmaras turbulence model, thus mitigating the potential errors arising from a mismatch of the adopted turbulence modeling through test simulations. However, this straightforward analytical approach neglects pressure gradients and is anticipated to be outperformed by the proposed model.

The evaluations are performed by evaluating the skin friction coefficient C_f and its normalized error introduced in equation (4.33).

4.5.2.2 Results

Figure 4.19 presents the results for the Wortmann FX60-100 airfoil at a 0° angle of attack, considering Reynolds numbers of $Re = 6 \cdot 10^6$ and 10^7 . For the sake of brevity, only

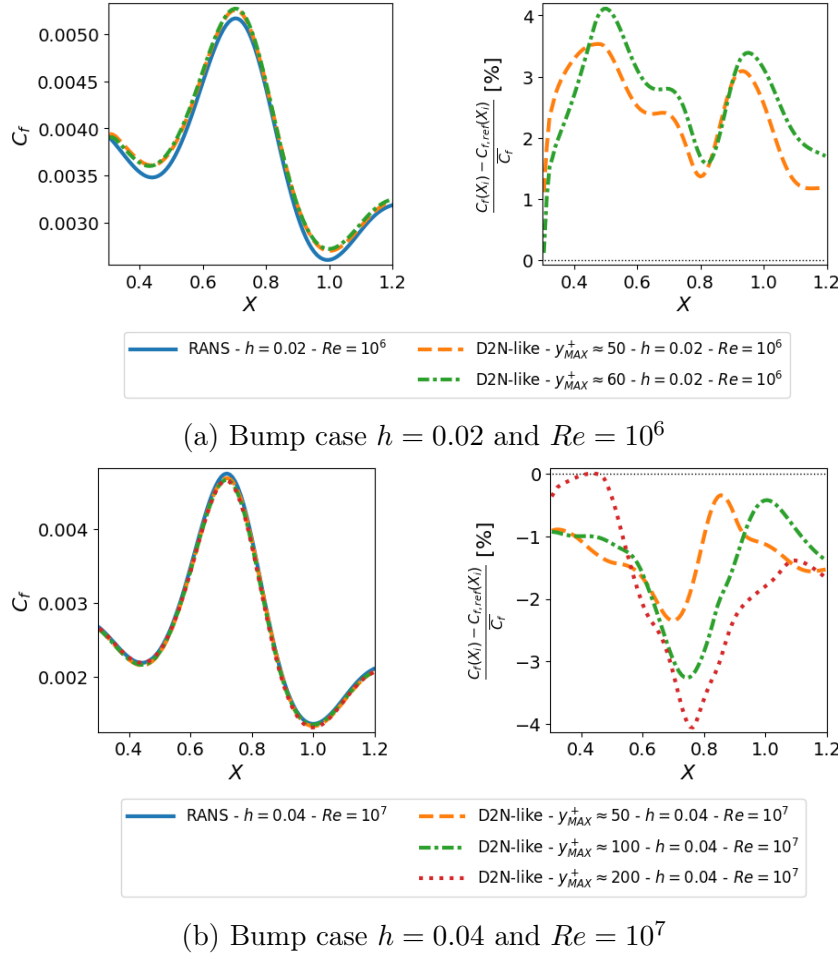


Figure 4.17: Skin friction coefficient C_f (left) and the relative local error (right) between the reference wall resolved simulation and wall modeled ones. Selected sample cases illustrating interpolation conditions across various bump heights with reference to training dataset. RANS-wall model interface placed at target $y_{MAX}^+ \approx 50, 100, 200$. When the target interface height does not respect $y/\delta < 0.15$, the interface is fixed at $y/\delta_{MAX} \approx 0.15$ and the obtained y_{MAX}^+ is shown in the legend.

the results along the suction side are presented, as the pressure side exhibits similar behavior. The figure displays the skin friction coefficient and its normalized error for both the wall-modeled simulation and the reference RANS simulation. The fully data-driven approach is compared with the analytical Spalart-Allmaras wall model. Both neural networks were applied for a local Reynolds number, Re_x , above $5 \cdot 10^5$ which sets the application point for the wall model at approximately the 8% and 5% of the chord, respectively, for the case at $Re = 6 \cdot 10^6$ and 10^7 .

The results demonstrate significantly higher accuracy with the fully data-driven wall model, which estimates the skin friction coefficient C_f for both flow configurations with an error consistently below 2% and maintaining near-constant behavior along the streamwise direction. Conversely, the Spalart-Allmaras wall law exhibits an error peaking at nearly 16% near the model application point, which gradually decreases along the airfoil, with a slight increase near the trailing edge. This behavior is well-explained by the evolution of the dimensionless pressure gradient on the airfoil: a strong favorable pressure gradient near the leading edge affects the boundary layer, the pressure gradient reaches near-zero values in the central portion of the airfoil and an adverse pressure gradient develops

RANS interface at $y_{MAX}^+ \approx 50$		
	$h = 0.02$	$h = 0.04$
$Re = 2 \cdot 10^6$	1.17%	0.96%
$Re = 4.5 \cdot 10^6$	0.54%	0.98%
$Re = 7 \cdot 10^6$	0.61%	1.03%
$Re = 9 \cdot 10^6$	0.88%	1.25%
RANS interface at $y_{MAX}^+ \approx 200$		
	$h = 0.02$	$h = 0.04$
$Re = 2 \cdot 10^6$	1.1%*	1.54%*
$Re = 4.5 \cdot 10^6$	0.95%*	1.17%
$Re = 7 \cdot 10^6$	1.07%	1.03%
$Re = 9 \cdot 10^6$	1.76%	1.66%

Table 4.8: 2-norm global error on the skin friction coefficient C_f between wall-modeled simulations and reference wall-resolved RANS for two different RANS-Wall model interface positions and different combinations of bump height and Reynolds number Re . Test case simulations representing interpolation condition on bump height ($h = 0.02$ and $h = 0.04$) and Reynolds number ($Re = 2 \cdot 10^6$, $4.5 \cdot 10^6$, $7 \cdot 10^6$ and $9 \cdot 10^6$). The simulations where the interface is located for y_{MAX}^+ lower than target value to satisfy $y/\delta \leq 0.15$ are marked by *. The actual interface height is shown in table 4.3.

approaching the trailing edge. This also explains the discrepancy in performance between the $Re = 6 \cdot 10^6$ and $Re = 10^7$ cases, with the former experiencing stronger dimensionless pressure gradients along the airfoil.

4.5.3 Numerical performances assessment

This section evaluates the numerical performance of the model. First, the computational cost of the fully data-based model is compared to that of the previous iterative approach from [92], using the analytical Spalart-Allmaras wall model as a reference. Then, the total convergence time for a flow over a flat plate is assessed by comparing a fully resolved RANS simulation to wall-modeled ones, using both the data-driven wall model and the Spalart-Allmaras wall law.

It should be noted that the computational performance is highly dependent on the solver configuration. The CFD solver used was not designed to efficiently integrate neural networks, and achieving optimal performance would require substantial modifications to the code structure that goes beyond the scope of the work. Therefore, the present results may be seen as an upper bound of computational costs. With appropriate optimizations and enhancements to the code, the computational time may surely be significantly reduced.

4.5.3.1 Model computational time assessment

The computational time of the proposed fully data-driven approach is compared with that of the iterative approach from [92]. For a fair comparison that is not biased by the reduced feature set of the iterative approach, both strategies are evaluated using the input features described in chapter 3 and the set from the current one. The network architecture considered are designed to be similar. In the first case, the dimensionless velocity u^+ is modeled as $f(y^+, p^+)$, with the near-wall Spalart-Allmaras model represented through linear evolution. In the second case, the dimensionless velocity u^+ is modeled using a wider

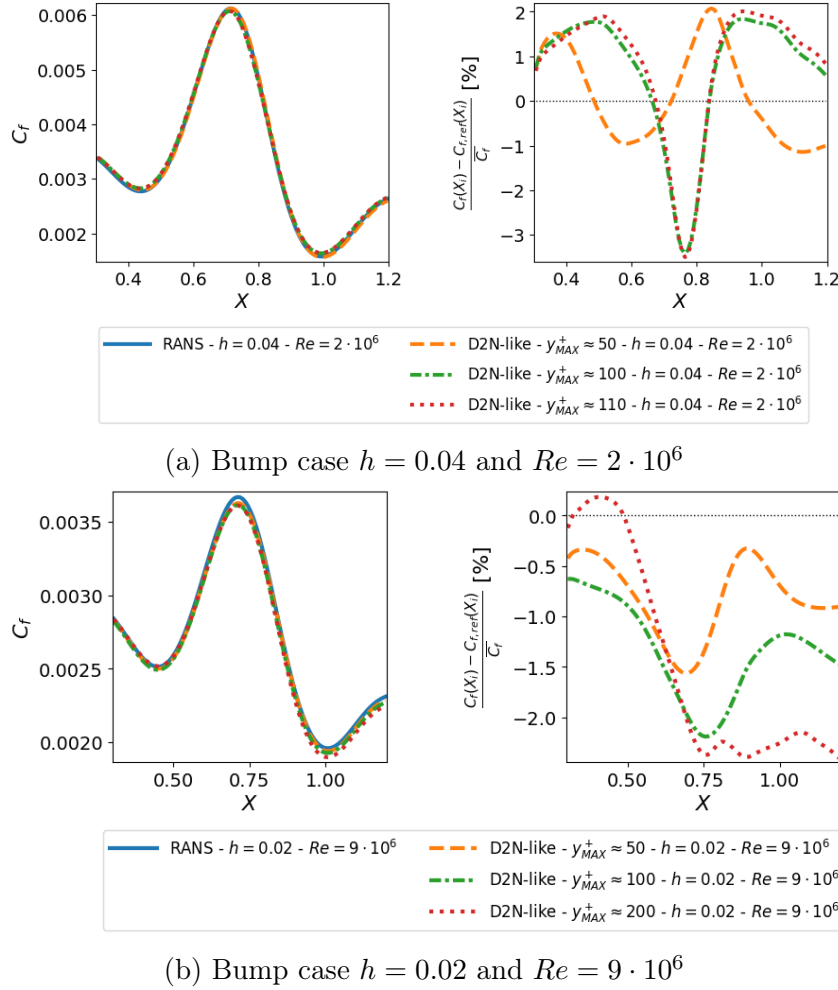


Figure 4.18: Skin friction coefficient C_f (left) and the relative local error (right) between the reference wall resolved simulation and wall modeled ones. Selected sample cases illustrating interpolation conditions across various bump heights and Reynolds numbers with reference to training dataset. RANS-wall model interface placed at target $y_{MAX}^+ \approx 50, 100, 200$. When the target interface height does not respect $y/\delta < 0.15$, the interface is fixed at $y/\delta_{MAX} \approx 0.15$ and the obtained y_{MAX}^+ is shown in the legend.

neural network that includes additional input features, such as $y^+, p^+, \partial p^+, \tilde{v}^+$, matching the fully data-driven approach presented. Here, the near-wall Spalart-Allmaras model is based on a polynomial regression for all cases and all strategies.

The evaluation is performed on the bump case, characterized by $h = 0.06$ and $Re = 6 \cdot 10^6$, considering different modeling distances of $y_{MAX}^+ \approx 10, 30, 50, 100$ and 200. The modeled region consists of 7, 18, 25, 36 and 48 layers of cells in the wall normal direction, respectively. The evaluated modeling strategy conforms to that of figure 2.7. Only ghost cells are effectively modeled, which allows for a reduction in computational time by neglecting useless cells during the simulation process. Two layers of cells are modeled for the upper RANS-Wall model boundary, and two layers of cells are modeled for the side boundary, with the modeling extending from the wall to the upper interface.

Figure 4.20 shows the CPU times required for a single application of the wall model with reference to the average computational cost of the Spalart-Allmaras wall law. The time per iteration is averaged over 1000 iterations and across all wall-modeled points in the simulation. Accordingly, the computational time of iterative processes is averaged

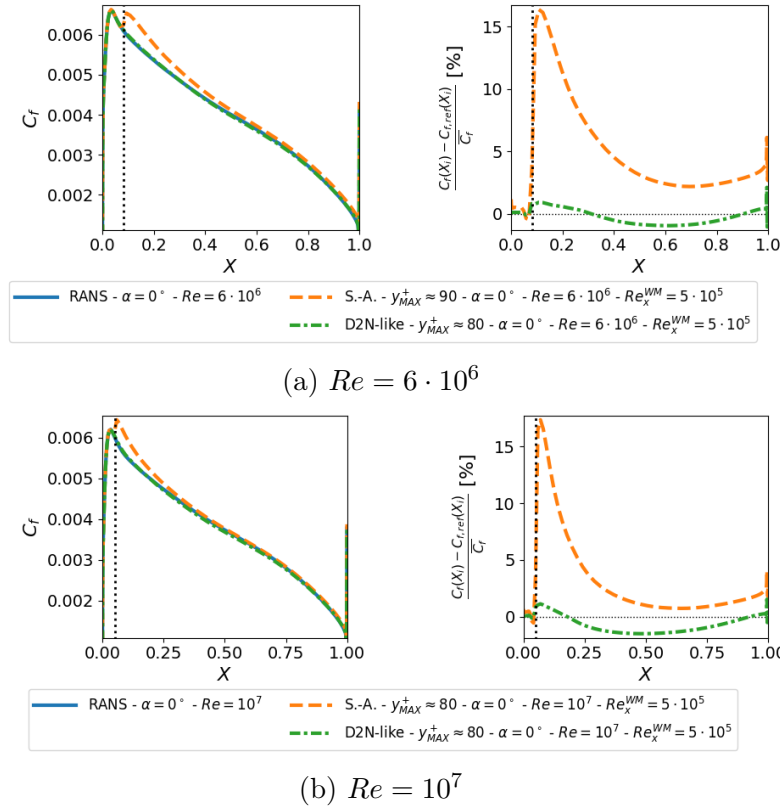


Figure 4.19: Flow over a Wortmann FX60-100 airfoil at $\alpha = 0^\circ$. Pressure side at $Re = 6 \cdot 10^6$ and $Re = 10^7$. Comparison between data-driven wall model and Spalart-Allmaras wall model. Skin friction coefficient C_f (left) and the relative local error (right) between the reference wall resolved simulation and wall modeled ones. The wall models are applied starting from the local Reynolds number $Re_x^{WM} = 5 \cdot 10^5$, which gives $X \approx 0.08$ and $X \approx 0.05$, respectively (dashed line). The RANS-wall model interface is placed at the wall distance giving $y/\delta_{MAX} \approx 0.15$ along the modeled section of the flow. The obtained y_{MAX}^+ is shown in the legend.

across all different RANS - Wall model interface distances.

Overall, the novel fully-data driven model is approximately 26 times more computationally expensive than the Spalart-Allmaras law. Furthermore, data driven approaches based on the strategy from [92] (that rely on iterative determination of local skin friction coefficients) are shown to be, on average, respectively 55 and 62 times more expensive than the analytical model. Even if all the deep-learning based approaches are sensibly more expensive than the analytical model, the present fully data-driven approach requires, on average, 26% less computational time than the iterative method from Romanelli et al. [92], presented in chapter 3, and 30% less than the iterative approach using the extended input feature set. It is important to remark that the three data-driven approaches show similar trends with different total computing time. This means that the primary difference in the required time is not due to the ghost cell filling process, requiring almost the same amount of computational time, but rather due to the determination of the local friction coefficient, which effectively is the main focus of the current work. On that aspect, our novel fully data-driven strategy is, on average, 5 times more computationally effective than the iterative method from [92].

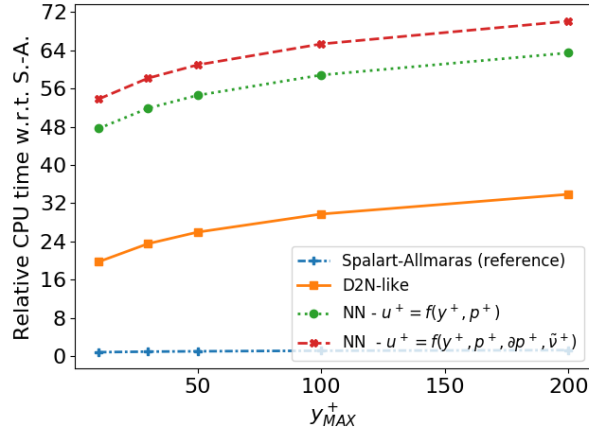


Figure 4.20: Average CPU time required by a single wall model application. Relative CPU time with reference to the mean computational cost of the Spalart-Allmaras law. Only the ghost cell are modeled. Comparison between iterative approach and full data-driven one. RANS-wall model interface placed at target $y_{MAX}^+ \approx 10, 30, 50, 100, 200$, giving respectively 7, 18, 25, 36 and 48 modeled layers.

4.5.3.2 Total convergence time assessment

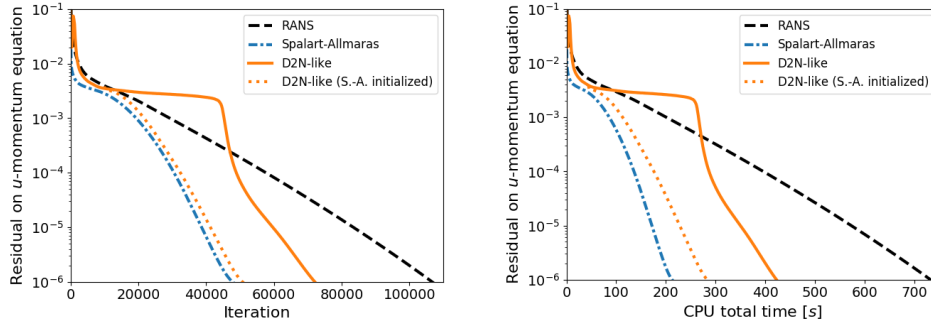
Here, the total convergence time for the fully data-driven method is compared both to a reference RANS simulation and a wall modeled simulation using the analytical Spalart-Allmaras wall law [110].

The selected case is that of a flow over a flat plate. The Reynolds number, based on a unitary length, is $5 \cdot 10^6$, and the Mach number is fixed at 0.2, with a free stream temperature fixed at $T_\infty = 300 K$. In order to strictly evaluate the convergence of the wall modeled flow, the domain does not include the leading edge of the flat plate. A zero-pressure boundary layer at a local Reynolds number of $Re_x = 2.5 \cdot 10^6$ is injected at the inlet of the domain, which extends to $Re_x = 10^7$ while a uniform flow field is used to initialize the simulation.

The computational domain is discretized by a uniform distribution of points in the streamwise direction, with 100 computational points used. In the wall normal direction, the first computational point is placed at a constant distance from the wall across all streamwise locations, ensuring a dimensionless wall distance below $y^+ = 0.2$ along the entire simulation domain. A growth ratio of 5% is applied to the cell size in the wall-normal direction to the maximum thickness of the boundary layer, expected for a $\delta = 0.03$, then a growth ratio of 15% is used outside the boundary layer. A computational grid of 100×172 is thus used.

Figure 4.21 shows the convergence of the momentum residuals along the streamwise direction for the reference fully resolved RANS simulation and the wall modeled ones, using both the Spalart-Allmaras analytical model and the fully data-driven model. Additionally, a wall-modeled simulation using the fully data-driven approach, initialized by 10'000 iterations using the Spalart-Allmaras wall law, as for the bump cases, is included.

The results demonstrate that, as expected, the Spalart-Allmaras law yields faster convergence, since the required computational time by the model is significantly reduced when compared to the data-driven approach. The latter, however, allows to reduce the computational time of more than 40% with 30% fewer iterations than the reference fully-resolved RANS simulation. It is important to highlight the slow convergence of the fully data-driven approach at the early stages of convergence. This is due to the



(a) Evolution of residual on the momentum equation in streamwise direction against the number of iteration. (b) Evolution of residual on the momentum equation in streamwise direction against total CPU time.

Figure 4.21: Numerical performance assessment of wall modeled simulations. Comparison between convergence rate of analytical wall model and fully data-driven model compared to a reference fully resolved RANS simulation. Convergence evaluated through the evolution of residual on the momentum equation in streamwise direction.

uniform flow initialization of the solution, which causes the neural networks to work in extrapolation condition. This could also lead to divergence issues for uniform flow initialization for cases presenting strong pressure gradients. This may be easily solved by applying the data-driven model in a second stage after some preliminary iterations with the Spalart-Allmaras law. This approach yields even faster convergence, with a reduction of computational time of about 60% in 50% of iterations.

4.6 Conclusion

This chapter introduced a new data-based wall model strategy for RANS simulations. The proposed model is inspired by the work discussed in chapter 3, but includes several new ideas that correct weaknesses of their previous approach. Both models aim to replace the resolution of the near-wall region by forcing the evolution of the primitive variables at the interface separating the RANS resolved region and the modeled region of the computational domain. In this approach, the modeled cells act as ghost cells for the resolved region, belonging to the numerical stencil of the first RANS resolved cell layers.

The proposed model is composed of three main components, modeling the near-wall thermodynamic state, the evolution of the velocity profile, and the near-wall behavior of the turbulent variable (the turbulence model considered for the article is the Spalart-Allmaras model).

The previous approach, from Romanelli et al. [92] and presented in the previous chapter, relies on a neural network to reconstruct the near-wall evolution of the dimensionless velocity u^+ as a function of the dimensionless wall distance y^+ and the dimensionless pressure gradient p^+ . The model employs an iterative approach to determine the local shear stress, involving, during the iterations, multiple inferences of the neural network. This is responsible for the high computational cost of the model. The chapter is thus devoted to propose a new fully data-driven approach, where a first neural network determines the local skin friction, effectively replacing the computationally expensive iterative process, and a second one models the velocity profile. However, without careful treatment, replacing

the iterative estimation of the wall shear stress with a direct data-based approach yields computational instabilities, potentially leading to convergence issues. The direct fully data-driven method does not ensure the continuity of the velocity profile as well as the previous iterative approach. To solve this problem, instead of reconstructing the dimensionless velocity profile, the second neural network is trained to estimate the wall normal derivative of the velocity, which effectively eliminates the discontinuity between the modeled and the resolved region of the wall-bounded region. The model can thus be formulated as a Dirichlet-To-Neumann map, which defines a function estimating the normal derivative of the velocity at the boundary between the RANS region and the near-wall modeled one.

Additional improvements have been proposed in the article. A new set of dimensionless features is introduced to improve the accuracy of the approach. Furthermore, the linear evolution assumed for the near-wall behavior of the Spalart-Allmaras variable from [92] proved to be inaccurate when the RANS - Wall model interface was far from the wall. The new approach introduces a polynomial regression model to enhance the accuracy of the Spalart-Allmaras variable at greater distances from the wall.

The data-driven approach of the proposed model consists of two neural networks trained on a dataset extracted from finely resolved RANS simulations of flow over a bump, under various Reynolds numbers and pressure gradient levels based on bump height. The model was tested and compared with fully resolved RANS simulations, using both training dataset cases and unseen configurations characterized by different combinations of Reynolds numbers and bump heights. The model demonstrated accurate results for all test cases, allowing for greater modeling distances from the wall and outperforming the previous approach both in terms of accuracy and required computational load. The new fully data-driven approach revealed to be approximately 26% less computationally expensive than the previous strategy, despite the more complex set of input features and modeling strategy for the Spalart-Allmaras near-wall behavior, and 30% less costly than the iterative approach fed with an equivalent set of input features. These results are due to the computationally more efficient estimation of the local skin friction, which is around five times less expensive than the iterative one. Finally, the robustness of the model was tested on a completely different flow configuration: the wall-bounded flow around an airfoil. The data-based model significantly outperformed an analytical wall model in terms of accuracy.

The data-driven wall model introduced in this study has demonstrated significant accuracy and cost-effectiveness compared to the previous approach. Furthermore, the inherent flexibility of the data-driven DtN framework, along with its ability to incorporate non-locality into the model, offers a compelling direction for future research. This could involve using multiple flow states, distributed along the streamwise direction, as inputs to reconstruct the boundary layer evolution over an extended portion of the near-wall region. Additionally, this framework could be extended to enhance data-driven modeling capabilities incorporating multiple components such as streamwise and wall-normal velocities, energy, density, or eddy-viscosity.

Additionally, the computational performance assessment has been done here with a rather preliminary, non-optimized implementation of the wall model in the solver. Further studies may be needed to evaluate more precisely and improve the computational cost of the present approach.

Finally, integrating this model into an Immersed Boundary Method framework could offer a promising opportunity to further test its robustness by tackling complex geometries, inspired by real-world industrial scenarios, in which traditional modeling techniques

struggle.

In conclusion, the proposed model provides an accurate and cost-effective alternative to conventional wall models, while also paving the way for future advancements in predictive tools for wall modeling.

Chapter 5

Current development and future works

5.1	Optimization of computational performance	125
5.2	Detection of extrapolation conditions	126
5.2.1	Density-based outliers detection	127
5.2.2	Outliers detection Gilbert–Johnson–Keerthi algorithm	127
5.2.3	Discussion of application	129
5.3	Extension of validity domain to complex flow configurations	130
5.4	Immersed Boundary Method application	131

This chapter focuses on the current research direction and future work perspectives. First, it introduces a pathway for improving the computational performance of the current model. Then, it discusses a method for detecting extrapolation conditions during model application. Moreover, potential improvements to the model are presented, in order to extend its applicability to complex flows, including flow separation, highlighting the approach to be followed to enhance the model’s capabilities. Finally, an overview of the challenges and issues encountered in a preliminary application of the model in an Immersed Boundary Method simulation is presented, presenting the actions and considerations needed to fully adapt the presented model to Cartesian solvers.

5.1 Optimization of computational performance

The computational performance assessment in this work was conducted using a non-optimized implementation of the wall model within the CFD solver. Although functional, the current setup may not fully capture the model’s potential efficiency, as bottlenecks could arise, especially in data handling, memory management or inference speed within the solver. Further investigations are recommended to refine this integration, focusing on optimizing data pipelines for neural network parameters, deeply modifying the data structures of the CFD code to more efficiently store neural network parameters and potentially allowing for a more streamlined integration.

Another promising avenue for enhancing the model’s performance is to utilize GPU-based inference. GPUs, designed for parallel processing, can significantly speed up neural network inference compared to traditional CPU computations. This approach would facilitate faster evaluations of the wall model during simulations, enabling the integration of more complex and larger neural networks without adversely affecting the overall computational time of the CFD analysis. Consequently, this would allow for the implementation of more sophisticated and refined models, thereby better addressing intricate flow configurations.

5.2 Detection of extrapolation conditions

The results have shown that the models are generally not able to extrapolate to input quantities that go beyond what was seen during training. While the trained model works well in interpolation conditions, it tends to fail when extrapolating. It is therefore crucial to be able to evaluate a priori if the model is used or not in its validity domain. In the following, two approaches to detect extrapolation conditions are explored and presented. Both the presented approaches are grounded in the concept of outlier detection, which is widely used in machine learning to identify data points that deviate significantly from the training distribution [46]. First, an outlier detection algorithm [111] based on the density of the training dataset is presented, then, it is followed by a more unconventional approach based on the Gilbert–Johnson–Keerthi (GJK) algorithm. These strategies can enable the detection of input features that lie outside the combination of parameters encountered during training, providing a safeguard against the neural network being forced to operate in extrapolation conditions, which can lead to simulation failures or non-convergence, as highlighted in chapter 3.

In order to provide a case of study, the algorithms are applied to the dataset of the neural network h presented in chapter 4. To enhance readability and interpretation of the results, the complete set of input features, initially consisting of four parameters, is reduced to two. Consequently, only the features η and β are considered. The evaluation of extrapolation conditions are then performed on a grid of 100×100 evaluation points, built within the range defined by the minimum and maximum values of each considered input feature. The figure 5.1 show the dataset and the grid of evaluation points built around it.

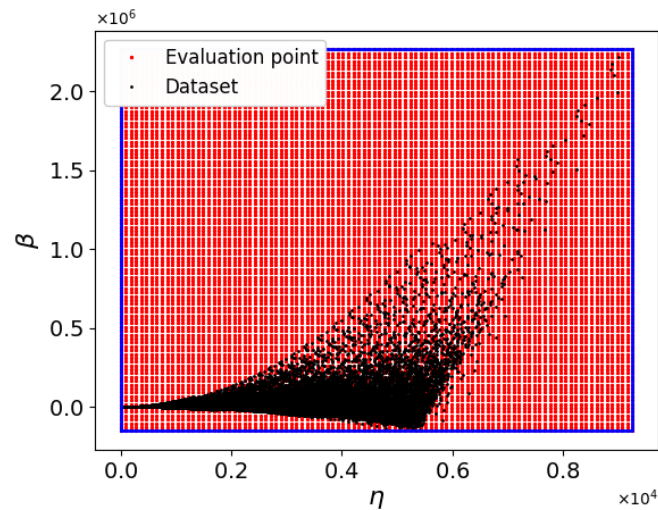


Figure 5.1: Dataset and evaluation points for outliers detection algorithms testing. Input features η and β are extracted from the simplified training dataset of neural network h , presented in chapter 4

It is important to note that detecting extrapolation conditions is straightforward when the desired input features fall outside the range given by the minimum and maximum value of each considered input feature in the training dataset. However, it can be possible, even inside this range, to have unseen combination of variables since due to a distribution of the training data not uniformly distributed, resulting in extrapolation conditions occurring between the minimum and maximum values of all the considered input feature.

5.2.1 Density-based outliers detection

The first approach to detect potential extrapolation condition is attempted through density-based outliers detection, as shown by Breunig et al. [13].

The approach aims to compute an outlier factor F as the ratio of the local dataset density, computed at the evaluation point and the average local density, computed among the dataset samples. In order to address the uneven distribution of training samples in the dataset, the computation of the outlier factor F is conducted locally, considering only the neighboring samples to the evaluation point. Consequently, given the set of evaluation samples $\hat{\mathbf{x}}$, for which the extrapolation or interpolation condition is to be estimated, and the N_k neighboring training samples $\tilde{\mathbf{x}}$, the outlier factor F for the i -th evaluation sample is computed as follows:

$$F(\hat{\mathbf{x}}_i, \tilde{\mathbf{x}}) = \frac{\rho(\hat{\mathbf{x}}_i; \tilde{\mathbf{x}})}{\frac{1}{N_k} \sum_{j=0}^{N_k} \rho(\tilde{\mathbf{x}}_j; \tilde{\mathbf{x}})}, \quad (5.1)$$

in which ρ is the local density of samples. It is computed as

$$\rho(\mathbf{x}_i, \mathbf{x}_k) = \frac{1}{N_k h} \sum_{j=0}^{N_k} K\left(\frac{\mathbf{x}_i - \mathbf{x}_j}{h}\right), \quad (5.2)$$

using a KDE technique, already presented in section 2.1.2.2.3. Here, Gaussian kernels of bandwidth h are employed for the density estimation. The evaluation of the outlier factor F and the local density is performed on standardized dataset obtained through equation (2.1).

An example of outlier factor F is presented in figure 5.2, obtained considering the reduced dataset, presented in chapter 4, using the grid of 100×100 evaluations points introduced before. In the evaluation, 10 neighboring samples to the evaluation point are considered and a bandwidth $h = 0.5$ is used. An outlier factor F close to unity indicates that the sample density at the evaluation point is similar to that of neighboring samples in the dataset. As a result, interpolation conditions are more likely when the combination of input features at the evaluation point is fed to a NN trained on this dataset. On the contrary, lower outlier factors indicated possible extrapolation conditions.

To identify the estimated outliers, a threshold for the outlier factor F must be established. In this case, a threshold of 0.99 is used.

This approach demonstrated promising results in terms of outlier detection accuracy; however, it was found to be highly dependent on both the number of neighbors considered and the kernel bandwidth, as illustrated in the figure 5.3.

5.2.2 Outliers detection Gilbert–Johnson–Keerthi algorithm

The Gilbert–Johnson–Keerthi (GJK) algorithm is a widely used method in computational geometry for detecting collisions or determining the minimum distance between two convex shapes [37], widely adopted in real-time applications like physics simulations, robotics or computer graphics due to its efficiency [116, 33].

The objective is to detect whether the evaluation point lies within the convex shape defined by the sample points in the training dataset. If the point is located inside this shape, the neural network trained on the dataset will operate under interpolation conditions when it encounters the input feature combination represented by the evaluation point. Conversely, if the evaluation point lies outside this convex shape, the neural network will be operating under extrapolation conditions.

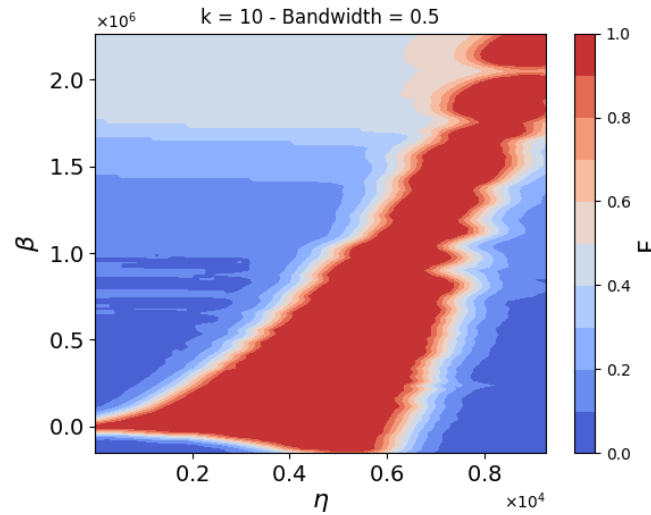


Figure 5.2: Outlier factor F for outlier detection considering 10 neighbors and a bandwidth $h = 0.5$ for the Gaussian kernels.

The GJK algorithm is based on the Minkowski difference operation [43], which is defined as an operation on subsets of a vector space:

$$A - B := \{a - b | a \in A, b \in B\}, \quad (5.3)$$

where A and B are position vectors representing points on the respective convex shapes under analysis for collision. In our case, one set is represented by the ensemble of sample points in the training dataset, while the other consists solely of the evaluation point.

The Minkowski difference defines, in Euclidean space, a shape that, in the event of a collision between the two sets of points, will contain the origin of the space under consideration. The algorithm thus evaluates whether the resulting Minkowski shape contains the origin. However, its efficiency comes from the fact that it does not compute the entire shape at once. Instead, it iteratively constructs and evaluates the simplex shapes [32] that compose the Minkowski difference, allowing for a more computationally efficient determination of collision or proximity.

These simplices are constructed by first selecting an arbitrary search direction and then choosing the point in each set that maximizes the dot product with that direction, effectively identifying the farthest point in that direction from the sample sets. Subsequently, the following search directions are chosen to ensure that the constructed simplices are as close as possible to the origin. If the initial simplex does not contain the origin, the search proceeds by constructing adjacent simplices in the direction of the origin of the Euclidean space, continuing until the origin is enclosed or the search concludes.

As introduced before, the GJK algorithm is adapted to convex sets of points. However, the dataset of samples can possibly not satisfy this condition, being enclosed in concave shape. To limit this problem, the algorithm is applied on portions of the dataset. Each portion is selected by considering the k neighboring samples to the evaluated point, minimizing local error. However, more sophisticated methods for dividing the dataset into overlapping convex shapes, as demonstrated by Chazelle [19], offer potential improvements. These alternative approaches, though promising, are left for future exploration.

In the figure 5.4, the application of the GJK algorithm is shown on the reduced dataset of the neural network h presented in chapter 4, using the grid of 100×100 evaluation

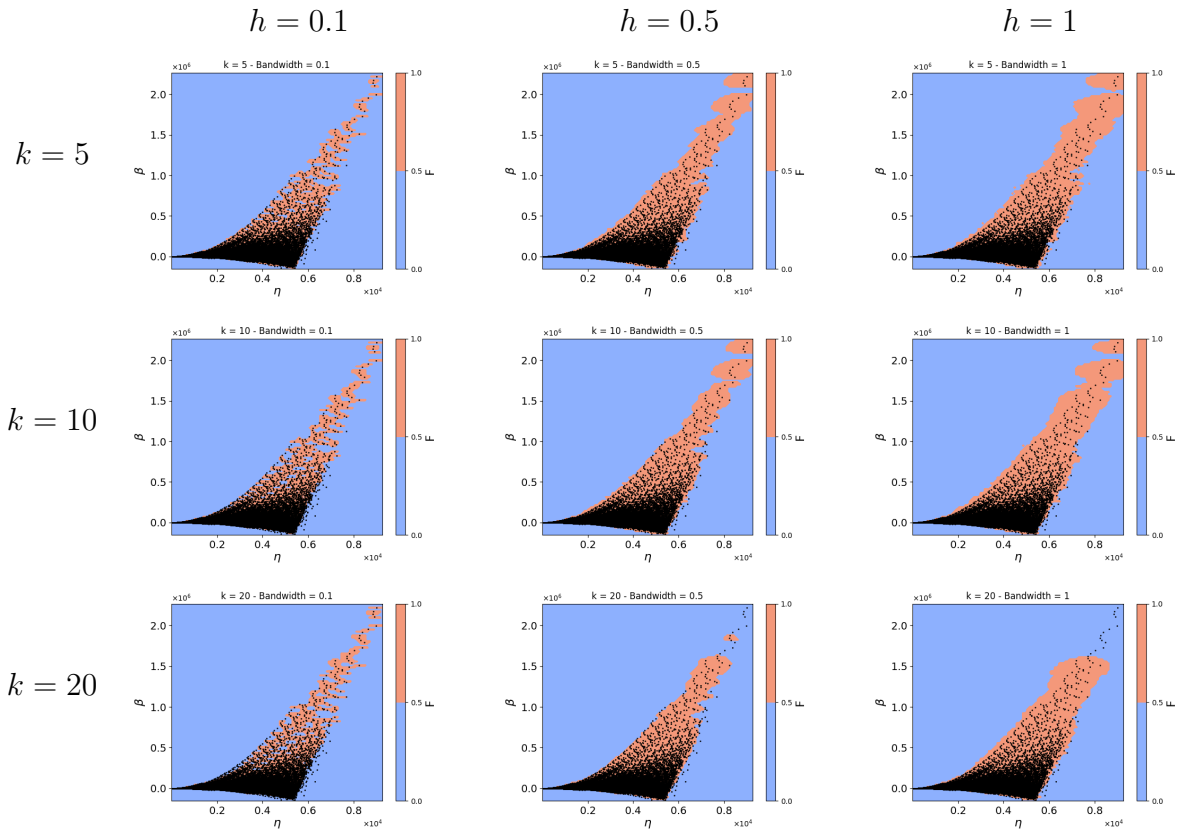


Figure 5.3: Application of density based algorithm for outlier detection. Comparison between different number of k neighbors ($k = 5, 10, 20$) and bandwidth of the Gaussian kernels ($h = 0.1, 0.5, 1$). The training dataset is shown in *black*. The region in the input feature space where interpolation condition are expected are in *red*, while extrapolation region are in *blue*.

points introduced before. The partition of the training dataset is done by considering the 20 closest neighboring samples of the dataset.

The GJK algorithm demonstrates promising accuracy in detecting the boundaries of the training dataset. However, some local imprecision remains. This issue could likely be addressed by improving the treatment of concave shapes through a more suitable partitioning of the dataset into convex ensembles, as discussed earlier.

5.2.3 Discussion of application

These two presented approaches allow to identify a set of input parameters outside the explored dataset during training. In order to use this analysis during the application of the model, the most straightforward approach would be to train a neural network to classify, as interpolation or extrapolation sample, the grid of samples, built within the range defined by the minimum and maximum values of each considered input feature.

In this way, these new predictive capabilities can be applied in two different ways. Firstly, during the application, they can provide feedback highlighting where extrapolation occurs for assessing the reliability of the results and managing the extrapolation in a more controlled manner, such as by applying an analytical extrapolation method (e.g., linear extrapolation) from neural network estimation at the closest sample configuration seen during training. Secondly, identifying extrapolation conditions can provide crucial

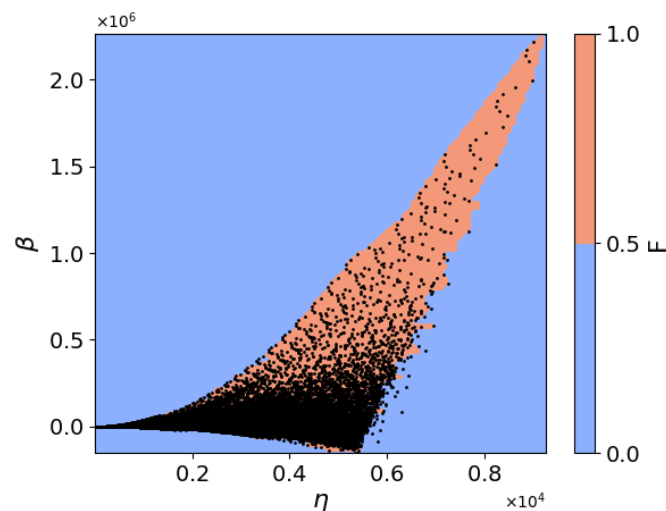


Figure 5.4: Application of Gilbert–Johnson–Keerthi (GJK) algorithm for outlier detection. The training dataset is shown in *black*. The region in the input feature space where interpolation condition are expected are in *red*, while extrapolation region are in *blue*.

information during the expansion of the training dataset. For instance, it allows for the selection of only the unseen configurations, enabling continuous retraining of the neural network and extending its domain of validity.

5.3 Extension of validity domain to complex flow configurations

The present work adapted and tested a data-driven wall model applied to quasi-equilibrium boundary layer, with moderate streamwise pressure gradient. A straightforward research direction for the presented methodology is to adapt and evaluate its performances on more complex flow configurations, such as boundary layer separation and reattachment flow.

In order to achieve this objective, an expansion of the training dataset is the first necessary step. Incorporating data from flows with strong adverse pressure gradients and separation will allow the model to generalize its predictions to more challenging scenarios. However, beyond merely expanding the dataset, a thorough study of this complex flows will help to eventually identify new input features capable to better capture key flow phenomena, which could be difficult to predict with the presented approach. This analysis would involve identifying the most critical flow variables and their interactions, ensuring that the neural network can accurately learn the underlying physics.

Moreover, careful consideration must be given to the numerical implementation of the wall model. In cases of flow separation, the placement of sampling points, used to recover data from the flow and drive the modeling, can significantly influence the results. Sampling points located within the recirculation bubble may respond differently than those placed above it, potentially leading to varied outcomes in the model’s predictions. This can highlight the need for adaptive sampling strategies that take into account the local flow characteristics to ensure accurate data representation.

Finally, modeling complex flow dynamics may require coupling the evolution of velocity in the near-wall region with the behavior of other primitive variables, essential for fully recovering the flow state and turbulence model response. The current approach couples the

data-driven velocity model with physical and regression-based analytical models for the thermodynamic state and the Spalart-Allmaras turbulence model variable. However, in the case of more complex flows, these analytical models may lack the generality needed to accurately capture near-wall behavior. A potential solution is to integrate the modeling of these variables directly into the neural networks used in the model. This fully data-driven approach could unlock the full potential of neural networks to capture intricate flow dynamics more effectively.

5.4 Immersed Boundary Method application

The methodology of the proposed model, designed to fully reconstruct the flow in the near-wall region and replace the resolution of the RANS equations, makes it particularly well-suited for use in Immersed Boundary Method (IBM) simulations. In IBM approach, in fact, the interaction between the flow and the boundary is imposed through forcing terms, since the physical wall is not represented in the computational domain. Wall models are therefore employed to impose the boundary layer development by replicating the effects of the wall on the flow.

However, the domain discretization in IBM simulations makes them less suitable for wall model development, as the Cartesian grids require a large number of cells due to uniform discretization in all spatial directions, resulting in longer computational times. Consequently, the development of the current wall model is carried out in body-fitted cases. Nevertheless, preliminary work has been undertaken to implement the proposed data-driven model within an IBM framework. In the detail, the data-driven approach in chapter 3 has been tested in an IBM simulation of the flow around the bi-dimensional bump.

Even though the results are encouraging, the performance in terms of accuracy of the model are far from what showed in body-fitted cases. This aspect arises from the additional numerical strategies needed to effectively address the IBM framework, as showed by Constant et al. [24].

An important consideration is the selection of force and forcing points. In IBM simulations, the wall model is employed to reconstruct the flow state in the wall-bounded region at a set of forcing points I , in order to replace the no-slip boundary condition at points P , located on the wall surface along the wall normal direction passing through the forced point. The forced points I , located at cell centers, are selected in order to completely fill the numerical stencil of resolved cells in the near wall region. In order to feed the employed model with required flow state data, forcing points S are needed farther from wall and they are located along the wall normal direction intersecting points I and P . A representative scheme of the application of wall model in an IBM is shown in figure 5.5.

This means that the location of forcing points is not guaranteed at cell centers and, consequently, interpolation techniques from adjacent computational cells are required in order to reconstruct the flow state fed to the model. It is thus clear that the choice of force and forcing points, as well as the interpolation technique and its stencil can have a huge impact on the wall model accuracy. As Capizzano [18] showed, inaccuracies in the numerical treatment of the boundary condition can lead to spurious oscillations in wall quantities. These oscillations can affect the accuracy and stability of the simulation, making it challenging to precisely capture key flow characteristics such as the shear stress at the wall. Methods to reduce these oscillations are thus needed. On this subject, methods for filtering out spurious fluctuations or averaging the flow state between multiple forcing

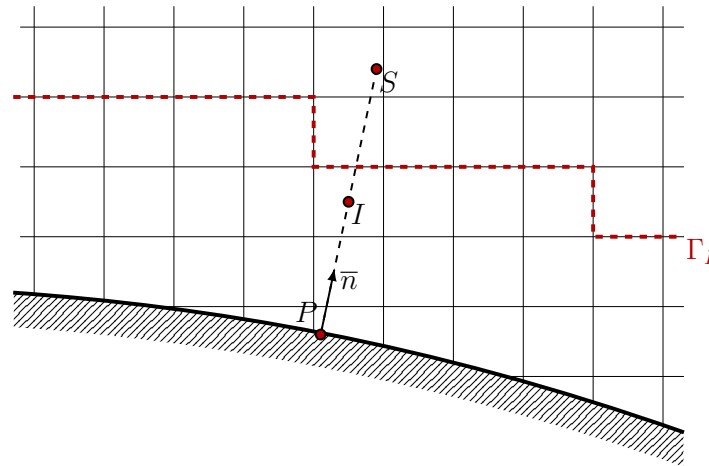


Figure 5.5: Representative scheme of a wall model (WM) application in an Immersed Boundary Method (IBM) framework.

points along the wall-normal direction have been introduced, as demonstrated by Constant [23].

An interesting direction for future research could be to adapt the presented wall model for applications within the IBM framework, incorporating the mentioned techniques to bridge the accuracy gap between the body-fitted applications shown in this work and the preliminary tests of the data-driven WM on Cartesian grids. This presents a promising opportunity to further assess the model's robustness by addressing complex geometries inspired by real-world industrial scenarios, where conventional modeling techniques encounter limitations.

Conclusion

The objective of this work was to develop a data-based wall model capable of fully reconstructing flow quantities in the near-wall region, providing greater accuracy and generality than traditional models. The present work is mainly methodological, and we have focused on body-fitted Reynolds Averaged Navier-Stokes (RANS) simulations, enabling rapid and lightweight simulations for testing different modeling strategies. Additionally, academic cases featuring diverse flow configurations and boundary layer evolutions were selected for the model's training and evaluation, ensuring a broad range of flow scenarios. The neural networks (NNs) used in this study are trained on fully resolved RANS simulations of turbulent flows over different two-dimensional bump geometries, considering various flow conditions through variations in Reynolds number and bump height. The performance of wall modeling strategies in terms of accuracy and computational cost has then been compared to fully resolved reference calculations for bump flow configurations not included in the training data, as well as an airfoil configuration.

In this work, a structured workflow was developed to construct a data-driven wall model, detailing each step from dataset extraction to model integration within the CFD solver. The process begins by extracting a dataset from RANS simulations. The data then undergoes through a preprocessing, consisting in scaling, normalizing and weighting the training samples. After selecting an optimal neural network architecture, the model is trained to replicate the development of the turbulent boundary layer. Once trained, the model's architecture and parameters are integrated into the CFD solver, enabling real-time boundary condition enforcement during simulations through embedded neural network inference. This workflow serves as a robust foundation for developing the presented data-driven wall models and offers a flexible tool for future advancements.

The proposed model aims to reconstruct the flow state in the region between the wall and a distant interface in the domain, transferring the wall boundary condition to an equivalent condition at the interface. The simulation domain is thus divided into a near-wall region, defined by a number of cell layers near the wall where the wall model is applied, and the rest, where RANS equations are solved. These two regions are separated by an interface that acts as a displaced boundary, with boundary conditions provided by the wall model. A ghost cell method is used to impose the boundary condition across this interface, modeling the flow in a series of ghost cells within the modeled domain region. Finally, applying the model requires knowing the local flow state, obtained through a sampling point located in the resolved region above the interface.

The presented wall law consists of three distinct components: a functional model describing the evolution of tangential velocity at the wall, based on a deep learning approach, a physical model governing the thermodynamic state and normal velocity at the wall and a model for turbulent viscosity in the near-wall region.

The initial approach developed in the thesis involves training a neural network on data resolved down to the wall to reconstruct dimensionless velocity profiles and model boundary layer evolution in the near-wall region. In line with analytical wall laws, velocity

is a function of distance from the wall and the pressure gradient, dimensioned by wall shear stress via a characteristic friction velocity, iteratively estimated using a Newton-Raphson algorithm. This method identifies the wall shear stress that ensures proper scaling of the relevant quantities at the sampling point to satisfy the functional provided by the neural network, ensuring the continuity of the modeled velocity profile across the interface. This method imposes a Dirichlet boundary condition for the RANS calculation. The modeling of the velocity profile is then sided by a physical model which fix the temperature through the Crocco-Busemann relation, linking velocity and temperature in the near-wall region. In the boundary layer, the pressure gradient in the wall-normal direction is neglected, allowing the density profile to be deduced using the ideal gas law by combining the reconstructed temperature and the pressure. Additionally, the wall-normal velocity is assumed to behave linearly in the modeled region. Finally, the Spalart-Allmaras turbulence model (S.-A.) model variable is reconstructed using a linear law based on the wall-normal distance dimensioned in the viscous sublayer and logarithmic layer.

This modeling approach produced promising results for almost all test cases. However, the model underestimated the wall friction coefficient, with increasing error as the interface height increased. This error was mainly due to growing deviation of the Spalart-Allmaras turbulence model from its modeling approach, as its linear behavior becomes increasingly erroneous as the distance from the wall and pressure gradients increase. Additionally, the iterative estimation of wall friction required multiple neural network inferences, leading to a high computational cost associated with model application.

To address the highlighted issues, an improved deep learning-based approach was developed. Inspired by the concept of the Dirichlet-to-Neumann (DtN) map, it directly imposes friction at the interface between the modeled region and the RANS calculation, thus fixing the normal velocity gradient at this interface and replacing the Dirichlet condition with a Neumann condition. The tangential velocity profile is then obtained by integrating the normal velocity gradient from its value at the interface to the wall. This method ensures the continuity of the velocity profile, even after replacing the iterative wall friction estimation. This is achieved using two interconnected neural networks: one predicts the wall shear from the friction velocity, and the other evaluates the near-wall normal velocity gradient. Changes were also made to the thermodynamic variables modeling. The Crocco-Busemann equation is still used, but pressure, wall-normal velocity, and the Spalart-Allmaras turbulence model variable are modeled via a polynomial regression. This approach ensures field continuity at the interface between the model and the resolved region while respecting the expected boundary conditions at the wall.

Overall, this approach demonstrated excellent robustness and accuracy, reproducing the reference friction coefficient with only a few percent errors for all studied cases, even with larger modeling distances, outperforming the previous approach in terms of both accuracy and computational cost. The new velocity profile modeling method, entirely based on deep learning, proved to be significantly less computationally expensive than the iterative strategy, allowing for a substantial reduction in computation time and the number of iterations compared to a fully resolved RANS simulation. This new approach was approximately 26% less computationally costly than the previous strategy, due to a more efficient calculation of local wall friction that was approximately five times less costly than the iterative approach.

The current results could serve as a starting point for further studies and investigations needed to overcome the challenges encountered in this work and extend the applicability of these deep learning-based wall models to more complex problems.

Firstly, the computational efficiency of the model can be improved. The computational

performance assessment was conducted with a non-optimized wall model implementation within the CFD solver, since it was not the priority of this work. Potential improvements include optimizing data pipelines and restructuring the data storage for NNs in the CFD solver, and exploring GPU-based inference for faster neural network evaluations. These refinements could significantly reduce computational time, enabling the integration of more complex models to address intricate flow configurations effectively.

Then, two promising approaches for detecting extrapolation conditions for NNs were explored: a density-based outlier detection algorithm and an unconventional approach using the Gilbert–Johnson–Keerthi (GJK) algorithm. Although these methods have already been implemented, further developed and assessment are needed to ensure that input features outside the training data’s parameter space are reliably and efficiently identified, preventing issues like non-convergence and improving the robustness of neural networks in complex simulations.

For future work, a first direction of research is to broaden the model’s applicability. This could involve more complex flow configurations, with stronger pressure gradients and flow separations. To achieve this, the training datasets should be expanded to include a wider range of such complex configurations. Additionally, a review of the neural network input variables might be required to better capture new complexities with more suitable input parameters.

Finally, a second research direction is to integrate this model into an Immersed Boundary Method (IBM) framework. The proposed modeling strategy, which provides a full representation of near-wall flow, thermodynamic state, and turbulence model required to close the RANS equations, is well-suited for IBM simulations. In this context, the interaction between the flow and the boundary is imposed through forcing terms, simulating the fluid-surface interaction within a Cartesian grid. Implementing this model for IBM simulations could offer a promising opportunity to further test the model’s robustness by tackling complex geometries inspired by real industrial scenarios, where traditional modeling techniques face limitations.

While these future research directions appear exciting, several crucial challenges remain. An important one is the balance between model complexity and accuracy, particularly in the intricate dynamics of boundary layers. The nonlinearity and fine-scale interactions in the near-wall region necessitate a careful architecture and hyperparameters optimization, as even minor prediction errors can magnify and lead to discrepancies from fully resolved simulation due to the high sensibility of turbulent boundary layers. Additionally, integrating the neural networks into the solver introduces stability challenges. Initial oscillations or nonphysical predictions may for instance disrupt convergence. To mitigate these issues, a focus on robust data preprocessing, targeted training strategies and a careful numerical implementation has to be considered.

Additionally, traditional RANS turbulence models frequently struggle to accurately capture specific physical phenomena within the resolved flow, such as junction or free shear flows. This limitation highlights the potential of NN-enhanced RANS closure models as a promising avenue for continued research. Although this work focuses on the near-wall region, data-driven wall models and NN-enhanced RANS models could complement each other effectively. Data-based wall models may significantly enhance the overall accuracy of RANS simulations when combined with NN-enhanced RANS closure models.

Finally, a similar work dedicated to LES simulations is definitively an important future research direction. The computational cost of neural network inference is less of an issue for data-driven wall models in high-fidelity simulations such as Large Eddy Simulation compared to the more cost-effective framework of body-fitted RANS. However, in the LES

environment, the modeling requirements are inherently more complex due to insufficient wall-normal resolution of the domain discretization and the coexistence with subgrid viscosity models. These aspects introduce additional challenges and issues that must be addressed for such applications.

French summary / Résumé en français

En mécanique des fluides numérique (CFD), la prédiction précise des écoulements turbulents en proche paroi est cruciale pour de nombreuses applications. La capacité à effectuer des simulations CFD rapides et précises est essentielle pour l'optimisation, l'évaluation des performances et les études paramétriques. Les simulations Reynolds Averaged Navier-Stokes (RANS) sont largement utilisées dans le secteur industriel en raison de leur faible coût de calcul. Toutefois, leur précision repose fortement sur la solution des écoulements en proche paroi nécessitant généralement un maillage très fin afin de saisir correctement les gradients importants présents dans la couche limite. Cela entraîne un impact significatif sur le coût de calcul de la simulation. Les lois de paroi permettent d'alléger ces calculs en substituant la résolution détaillée de la couche limite par une modélisation, ce qui permet de réduire les contraintes sur le maillage et de diminuer les temps de calcul.

De plus, dans le cas de la méthode des frontières immergées (IBM), l'utilisation des modèles de paroi devient essentielle, car cette méthode ne représente pas explicitement la paroi dans le maillage, représentée par une grille cartésienne. L'interaction entre l'écoulement et la frontière est imposée par un terme de forçage, ce qui rend les modèles de paroi nécessaires pour reproduire correctement les effets de la paroi.

Les modèles de paroi nécessitent des données provenant d'écoulement à une certaine hauteur au-dessus de la paroi, appelée point d'échantillonnage. Ce point constitue une interface entre l'écoulement résolu, éloigné de la paroi et l'écoulement modélisé, plus proche. À ce point, l'état de l'écoulement est extrait pour fournir les conditions aux limites nécessaires à la modélisation des contraintes de cisaillement, de l'évolution de la vitesse et de l'état thermodynamique dans la région en proche paroi, garantissant ainsi la cohérence entre le modèle de paroi et l'écoulement extérieur résolu.

Les approches de modélisation de paroi se divisent principalement en deux catégories : les méthodes hybrides et les modèles de contrainte de cisaillement.

Les méthodes hybrides séparent la région de l'écoulement résolu de celle modélisée par une interface, permettant ainsi de réduire le raffinement du maillage près de la paroi tout en maintenant la capture des phénomènes physiques essentiels. Les modèles de paroi hybrides sont souvent complexes et coûteux à mettre en œuvre, car ils nécessitent la résolution d'équations différentielles dans l'espace, généralement dérivées des équations de Navier-Stokes simplifiées.

En revanche, les modèles de contrainte de cisaillement étendent la région résolue jusqu'à la surface de la paroi, remplaçant la condition classique de non-glissement par une contrainte de cisaillement imposée. Les modèles de contrainte de cisaillement, moins exigeants en calcul, se divisent en deux types : les modèles analytiques, qui calculent directement la contrainte de cisaillement à partir de la vitesse à un point d'échantillonnage, et les modèles intégrales, qui dérivent des équations RANS simplifiées en intégrant l'équation de la quantité de mouvement de la paroi jusqu'au point d'échantillonnage.

Dans le contexte des lois de paroi, les approches basées sur l'apprentissage profond commencent à être explorées. La littérature récente a montré un intérêt croissant pour la modélisation de paroi basée sur les données, en particulier pour ce qui concerne les approches modélisant la contrainte de cisaillement à la paroi. Ces modèles cherchent principalement à pallier les limitations des approches traditionnelles en exploitant les réseaux de neurones pour améliorer la précision et gérer des phénomènes d'écoulement complexes. Bien que diverses stratégies d'apprentissage automatique et de sélection des données aient été explorées, les recherches sont encore incomplètes et une méthodologie commune n'est pas encore établie. Néanmoins, la flexibilité des réseaux de neurones offre un potentiel indéniable dans la modélisation des écoulements en proche paroi.

Ce travail a pour objectif de développer et évaluer une loi paroi basée sur l'apprentissage profond, capable de reproduire avec précision l'évolution de la région interne de la couche limite, fournissant ainsi les conditions aux limites pour le calcul RANS qui se déroule loin de la paroi.

Le modèle a pour objectif de reconstruire l'état de l'écoulement dans la région située entre la paroi et une interface distante dans le domaine, afin de transférer la condition limite de la paroi vers une condition équivalente sur l'interface. Dans un cadre numérique, ce modèle remplace le calcul de la solution proche de la paroi par une évolution modélisée. Le domaine de simulation est ainsi divisé en une région proche de la paroi, étant définie par un certain nombre de couches de cellules près de la paroi, où le modèle de paroi est utilisé, et le reste, où la résolution des équations RANS est requise. Ces deux zones sont séparées par une interface qui agit comme une frontière déplacée, avec des conditions limites définies par le modèle de paroi. Une méthode de cellules fantômes est utilisée pour imposer la condition limite à travers cette interface. Cela implique la modélisation de l'écoulement dans une série de cellules fantômes situées dans la région modélisée du domaine. Le nombre de cellules fantômes dépend du schéma numérique adopté, avec deux couches requises dans cette étude. De plus, l'application du modèle nécessite de connaître l'état local de l'écoulement, obtenu à travers un point d'échantillonnage situé dans la région résolue au-dessus de l'interface.

La loi de paroi se décompose en trois éléments distincts : un modèle fonctionnel décrivant l'évolution de la vitesse tangentielle au mur, basé sur une approche fondée sur les données ; un modèle physique qui gouverne l'état thermodynamique et le champ de vitesse normal au mur ; et un modèle pour la viscosité turbulente dans la région proche de la paroi.

Le développement et l'application d'un modèle de paroi basé sur les données dans un solveur CFD se déroulent en deux phases. Une première phase consiste à construire et entraîner le modèle, en commençant par la création d'un jeu de données dérivé de simulations RANS comprenant l'évolution de la couche limite sous différentes conditions d'écoulement. Ces données sont ensuite prétraitées pour préparer l'apprentissage automatique, incluant normalisation et la pondération de chaque échantillon. Un modèle d'apprentissage approprié est ensuite sélectionné et entraîné à reproduire la dynamique de la couche limite. La deuxième phase consiste à intégrer le modèle, donc le ou les réseaux de neurones dans le solveur CFD, afin de pouvoir l'appliquer lors des calculs. Les réseaux de neurones et les tâches d'apprentissage automatique sont pris en charge par la bibliothèque TensorFlow, tandis que le solveur de CFD utilisé est FAST Structured (FastS), développé par l'ONERA. Ce dernier a été modifié pour permettre l'exécution d'applications de réseaux de neurones pendant le calcul CFD.

Les réseaux neuronaux utilisés au cours de l'étude sont entraînés à partir de simulations RANS entièrement résolues d'écoulements turbulents sur différentes géométries de bosses

bidimensionnelles, prenant en compte diverses conditions d'écoulement à travers la variation du nombre de Reynolds et la hauteur de la bosse.

Les performances en termes de précision et coût computationnel des stratégies de modélisation pariétale sont ensuite comparés à des calculs de référence résolus jusqu'à la paroi pour des configurations d'écoulement sur une bosse non incluses dans les données d'entraînement, ainsi que sur une configuration de profil d'aile.

Une approche préliminaire consiste à entraîner un réseau neuronal sur des données résolues jusqu'à la paroi, afin de reconstruire des profils de vitesse adimensionnés et de modéliser l'évolution de la couche limite en proche paroi. Conformément aux lois de paroi analytiques, la vitesse est fonction de la distance à la paroi et du gradient de pression, ces variables étant adimensionnées par rapport au cisaillement visqueux pariétal via une vitesse de frottement caractéristique, estimée itérativement à l'aide d'un algorithme de Newton-Raphson. Cela permet d'identifier la contrainte pariétale qui garantit le correct adimensionnement des grandeurs concernées au point d'échantillonnage, afin de satisfaire le fonctionnel fourni par le réseau neuronal, ce qui garantit la continuité du profil de vitesse modélisé à travers l'interface. Cette méthode impose une condition aux limites de type Dirichlet pour le calcul RANS.

Les profils de température dans la portion modélisée du domaine et le long de la frontière sont modélisés à l'aide de la relation de Crocco-Busemann, adaptée aux conditions de paroi adiabatique. Cette relation établit un lien entre la vitesse et la température dans la région proche de la paroi. Dans la couche limite, le gradient de pression dans la direction normale à la paroi est négligé. Cela permet de déduire le profil de densité via la loi des gaz parfaits, en combinant la température reconstruite avec la loi de Crocco-Busemann et la pression estimée constante. De plus, la vitesse normale à la paroi est supposée se comporter de manière linéaire dans la région modélisée. Enfin, le comportement de la variable du modèle Spalart-Allmaras turbulence model (S.-A.) dans la région proche de la paroi est reconstruite à travers une loi linéaire, fonction de la distance à la paroi adimensionnée dans la sous-couche visqueuse et la couche logarithmique.

Le modèle a été testé et comparé à des simulations RANS entièrement résolues. Les cas tests ont été sélectionnés à la fois à partir du jeu de données d'entraînement et de configurations inédites de l'écoulement autour de la bosse, caractérisées par une combinaison différente du nombre de Reynolds et de hauteur. Cette approche de modélisation produit des résultats précis pour presque tous les cas tests et distance de modélisation testés. Cependant, le modèle sous-estime le coefficient de frottement de paroi, avec une erreur qui augmente à mesure que la hauteur de l'interface augmente. Cette erreur est principalement due à une déviation croissante dans le comportement du modèle de turbulence et de la variable de Spalart-Allmaras turbulence model. En effet, son comportement linéaire devient de plus en plus incorrect à mesure que l'on s'éloigne de la paroi et en présence de gradients de pression significatifs. De plus, l'estimation itérative du frottement pariétal nécessite plusieurs inférences du réseau de neurones, ce qui entraîne un coût computationnel élevé lié à l'application du modèle.

Pour réduire le coût associé à l'estimation itérative du cisaillement visqueux, une nouvelle approche basée sur l'apprentissage profond a été développée. Inspirée du concept de la Dirichlet-To-Neumann (DtN) map, elle impose directement le frottement à l'interface entre la région modélisée et le calcul RANS, fixant ainsi la dérivée normale du champ de vitesse à cette interface, en remplaçant la condition de Dirichlet par une condition de Neumann. Le profil de vitesse tangentielle est ensuite obtenu en intégrant la dérivée de la vitesse dans la direction normale, depuis sa valeur à l'interface jusqu'à la paroi. Cette méthode garantit la continuité du profil de vitesse, même après le remplacement de

l'estimation itérative du frottement pariétal.

La dérivée du champ de vitesse est estimée à l'aide de deux réseaux neuronaux interconnectés: l'un estime le cisaillement visqueux pariétal à partir de la vitesse de frottement, et l'autre évalue la dérivée normale du champ de vitesse proche de la paroi. Plus précisément, le premier réseau est entraîné pour prédire la distance adimensionnée de la paroi en fonction d'un ensemble de paramètres d'entrée issus de combinaisons adimensionnées de grandeurs physiques locales de l'écoulement. Cela permet par la suite d'estimer la vitesse caractéristique de frottement en connaissant la distance de la paroi du point d'échantillonnage. Le deuxième réseau permet l'évaluation de la dérivée normale à la paroi du champ de vitesse à travers un jeu de paramètres d'entrée qui reprend l'adimensionnement classique basé sur la contrainte visqueuse pariétale.

Des changements ont également été effectués au niveau de la modélisation des variables thermodynamiques de l'écoulement. L'équation de Crocco-Busemann est toujours utilisée, mais la modélisation de la pression, de la vitesse normale à la paroi et de la variable du modèle Spalart-Allmaras turbulence model est réalisée par une régression polynomiale. Cette approche garantit la continuité du champ au niveau de l'interface entre le modèle et la zone résolue, tout en respectant les conditions limites attendues à la paroi. Concernant la pression, un gradient nul est imposé à la paroi, tandis que pour la vitesse normale, à la fois le gradient normal et la vitesse sont nuls. Pour la variable de Spalart-Allmaras turbulence model, une valeur nulle est imposée à la paroi, et la pente de l'ancienne approche linéaire est conservée dans la région proche de la paroi.

Ce dernier modèle est testé sur des configurations différentes de bosse bidimensionnelle. Globalement, cette approche démontre une excellente robustesse et précision, reproduisant le coefficient de frottement de référence avec une erreur de seulement quelques pourcents pour tous les cas étudiés même avec des distances de modélisations plus grandes, en surpassant l'approche précédente à la fois en termes de précision et de coût computationnel.

La nouvelle méthode de modélisation du profil de vitesse, entièrement fondée sur l'apprentissage profond, s'est avérée nettement moins coûteuse que la stratégie itérative, tout en permettant une réduction significative du temps de calcul et une diminution du nombre d'itérations par rapport à une simulation RANS entièrement résolue. Cette nouvelle approche s'est révélée environ 26% moins coûteuse en calcul que la stratégie précédente, malgré un ensemble d'entrées et une stratégie de modélisation plus complexes pour le comportement proche de la paroi du modèle de Spalart-Allmaras turbulence model, et 30% moins coûteuse que l'approche itérative avec un ensemble d'entrées équivalent. Ces résultats s'expliquent par une estimation plus efficace en calcul du frottement pariétal local, environ cinq fois moins coûteuse que l'approche itérative.

Enfin, la robustesse du modèle a été testée sur une configuration d'écoulement complètement différente : l'écoulement autour d'un profil d'aile. Le modèle basé sur l'apprentissage profond a surpassé de manière significative un modèle de paroi analytique en termes de précision.

Les résultats actuels peuvent constituer un point de départ pour des études et investigations supplémentaires nécessaires afin de surmonter les problèmes rencontrés dans ce travail et d'étendre le domaine de validité de ces modèles de paroi basés sur l'apprentissage profond à des problèmes plus complexes.

Premièrement, l'efficacité computationnelle du modèle pourrait être améliorée. L'évaluation des performances en termes de coût computationnel a été réalisée avec une implémentation non optimisée du modèle de paroi dans le solveur CFD. Parmi les améliorations potentielles figurent l'optimisation des flux de données et la restructuration du stockage des données concernant les réseaux neuronaux au sein du solveur CFD, ainsi

que l'exploration de l'inférence basée sur GPU pour accélérer les évaluations du réseau de neurones. Ces ajustements pourraient réduire considérablement le temps de calcul, permettant ainsi l'intégration de modèles plus sophistiqués pour traiter efficacement des configurations d'écoulement plus complexes.

De plus, deux approches prometteuses pour détecter les conditions d'extrapolation des réseaux neuronaux ont été explorées : un algorithme de détection d'outliers basé sur la densité et une approche non conventionnelle utilisant l'algorithme Gilbert–Johnson–Keerthi (GJK). Bien que ces méthodes aient déjà été implémentées, un développement supplémentaire est nécessaire pour s'assurer que les paramètres d'entrée situés en dehors de l'espace des paramètres des données d'entraînement soient identifiées de manière fiable, afin de prévenir des problèmes tels que la non-convergence et d'améliorer la robustesse des neural networks dans des simulations complexes.

Pour des travaux futurs, une première direction d'étude consiste à élargir le domaine d'applicabilité du modèle. Cela pourrait inclure des configurations d'écoulement plus complexes, avec des gradients de pression plus importants et des séparations d'écoulement. Pour cela, une extension des ensembles de données d'entraînement devrait être prévue afin d'inclure un large éventail de ces configurations complexes. En second lieu, une étude des variables d'entrée des réseaux neuronaux pourrait être nécessaire pour mieux capturer les nouvelles complexités avec des paramètres d'entrée mieux adaptés.

Enfin, une deuxième direction d'étude consiste à intégrer ce modèle dans un cadre de méthode des frontières immergées (IBM). La stratégie de modélisation proposée, qui prévoit une modélisation complète de l'écoulement proche des parois, de son état thermodynamique, ainsi que du modèle de turbulence nécessaire à la fermeture des équations RANS, s'adapte bien aux calculs IBM. Dans ce cadre, l'interaction entre l'écoulement et la frontière est imposée par des termes de forçage, reproduisant les interactions du fluide avec la surface solide au sein d'un maillage cartésien. Une implémentation pour simulations IBM pourrait offrir une opportunité prometteuse pour tester davantage la robustesse du modèle en abordant des géométries complexes, inspirées de scénarios industriels réels, où les techniques de modélisation traditionnelles montrent leurs limites.

Bibliography

- [1] N. Afzal. “Power law and log law velocity profiles in fully developed turbulent pipe flow: Equivalent relations at large Reynolds numbers”. In: *Acta Mechanica* 151.3-4 (2001), pp. 171–183 (cit. on pp. 32, 64, 90).
- [2] N. Alferez et al. “Intel Xeon and Xeon Phi Optimizations of an Industry-Oriented Computational Fluid Dynamics Solver”. In: *Proceedings of the Intel HPC Developer Conference*. 2017, pp. 12, 16, 30, 45 (cit. on p. 53).
- [3] S. R. Allmaras, F. T. Johnson, and P. R. Spalart. “Modifications and Clarifications for the Implementation of the Spalart-Allmaras Turbulence Model”. In: *7th International Conference on Computational Fluid Dynamics (ICCFD7)*. Big Island, Hawaii, 2012 (cit. on p. 16).
- [4] Gonzalo Arranz et al. “Building-block-flow computational model for large-eddy simulation of external aerodynamic applications”. In: *Communications Engineering* 3.1 (2024), p. 127. URL: <https://doi.org/10.1038/s44172-024-00278-1> (cit. on p. 33).
- [5] Hyunseut J. Bae and Petros Koumoutsakos. “Scientific multi-agent reinforcement learning for wall-models of turbulent flows”. In: *Nature Communications* 13.1 (2022), p. 1443 (cit. on p. 33).
- [6] Elias Balaras, Carlo Benocci, and Ugo Piomelli. “A two-layer approximate boundary condition for large-eddy simulations”. In: *AIAA Journal* 34.6 (1996), pp. 1111–1119 (cit. on p. 30).
- [7] Yoshua Bengio. “Practical recommendations for gradient-based training of deep architectures”. In: *CoRR* abs/1206.5533 (2012). URL: <http://arxiv.org/abs/1206.5533> (cit. on p. 47).
- [8] Christophe Benoit et al. *Cassiopee (CFD Advanced Set of Services In an Open Python EnvironmEnt): A CFD pre and post-processing python package*. 2023. URL: <https://cassiopee.onera.fr/> (cit. on p. 53).
- [9] Sanjeeb T Bose and George I Park. “Wall-modeled large-eddy simulation for complex turbulent flows”. In: *Annual Review of Fluid Mechanics* 50 (2018), pp. 535–561 (cit. on pp. 29, 30).
- [10] Léon Bottou and Olivier Bousquet. “The Tradeoffs of Large Scale Learning”. In: *Optimization for Machine Learning*. Ed. by Suvrit Sra, Sebastian Nowozin, and Stephen J. Wright. Cambridge: MIT Press, 2012, pp. 351–368 (cit. on p. 39).
- [11] Margaux Boxho. “Development of machine learning-based wall shear stress models for LES in the presence of adverse pressure gradients and separation”. PhD thesis. Université de Liège, 2021. URL: <https://orbi.uliege.be/handle/2268/262621> (cit. on pp. 30, 31).

- [12] Lori A. Breslow and David W. Aha. “Simplifying Decision Trees: A Survey”. In: *The Knowledge Engineering Review* 12.1 (1997), pp. 1–47 (cit. on p. 40).
- [13] Markus M. Breunig et al. “LOF: identifying density-based local outliers”. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. SIGMOD '00. Dallas, Texas, USA: Association for Computing Machinery, 2000, pp. 93–104. URL: <https://doi.org/10.1145/342009.335388> (cit. on p. 127).
- [14] S. L. Brunton, B. R. Noack, and P. Koumoutsakos. “Machine learning for fluid mechanics”. In: *Annual Review of Fluid Mechanics* 52 (2020), pp. 477–508 (cit. on p. 33).
- [15] Arthur E. Bryson. “A Gradient Method for Optimizing Multi-Stage Allocation Processes”. In: *Proceedings of the Harvard Univ. Symposium on Digital Computers and Their Applications*. Cambridge: Harvard University Press, 1962, pp. 3–6 (cit. on p. 37).
- [16] E. Buckingham. “On physically similar systems; illustrations of the use of dimensional equations”. In: *Physical Review* 4.4 (1914), pp. 345–376 (cit. on p. 25).
- [17] Yao Cai, Zhongkui Li, and Jieping Ye. “Physics-Informed Neural Networks for Fluid Dynamics”. In: *Physical Review Letters* 124 (2020), p. 078001 (cit. on p. 34).
- [18] Francesco Capizzano. “Turbulent Wall Model for Immersed Boundary Methods”. In: *AIAA Journal* 49.11 (2011), pp. 2367–2381. URL: <https://doi.org/10.2514/1.J050466> (cit. on p. 131).
- [19] Bernard Chazelle. “Convex partitions of polyhedra: A lower bound and worst-case optimal algorithm”. In: *SIAM Journal on Computing* 13.3 (1983), pp. 488–507 (cit. on p. 128).
- [20] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. “Fast and accurate deep network learning by exponential linear units (ELUs)”. In: *arXiv preprint arXiv:1511.07289* (2015). URL: <https://arxiv.org/abs/1511.07289> (cit. on p. 35).
- [21] J. Cliquet, R. Houdeville, and D. Arnal. “Application of Laminar-Turbulent Transition Criteria in Navier-Stokes Computations”. In: *AIAA Journal* 46.5 (2008), pp. 1182–1190. URL: <https://doi.org/10.2514/1.30215> (cit. on p. 69).
- [22] Donald Coles. “The law of the wake in the turbulent boundary layer”. In: *Journal of Fluid Mechanics* 1.2 (1956), pp. 191–226 (cit. on p. 27).
- [23] Benjamin Constant. “Amélioration d’une méthode de frontières immergées pour la simulation d’écoulements turbulents autour de géométries complexes”. Theses. Université de Bordeaux, 2023. URL: <https://theses.hal.science/tel-04146884> (cit. on p. 132).
- [24] Benjamin Constant et al. “An improved Immersed Boundary Method for turbulent flow simulations on Cartesian grids”. In: *Journal of Computational Physics* 435 (2021), p. 110240. URL: <https://hal.archives-ouvertes.fr/hal-03182402> (cit. on p. 131).
- [25] Haskell B. Curry. “The Method of Steepest Descent for Non-linear Minimization Problems”. In: *Quarterly of Applied Mathematics* 2.3 (1944), pp. 258–261 (cit. on p. 39).

- [26] George Cybenko. “Approximation by Superpositions of a Sigmoidal Function”. In: *Mathematics of Control, Signals, and Systems* 2.4 (1989), pp. 303–314 (cit. on p. 35).
- [27] S. Deck. “Simulation numérique des charges latérales instationnaires sur des configurations de lanceur”. PhD thesis. Université d’Orléans, 2002 (cit. on p. 16).
- [28] TensorFlow Developers. *TensorFlow*. Version v2.15.1. 2024. URL: <https://doi.org/10.5281/zenodo.10798587> (cit. on pp. 46, 102).
- [29] Daniel Di Domenico, João V. F. Lima, and Gerson G. H. Cavalheiro. “NAS Parallel Benchmarks with Python: a performance and programming effort analysis focusing on GPUs”. In: *The Journal of Supercomputing* 79.8 (2023), pp. 8890–8911. URL: <https://doi.org/10.1007/s11227-022-04932-3> (cit. on p. 53).
- [30] Karthik Duraisamy, Gianluca Iaccarino, and Heng Xiao. “Perspectives on machine learning-augmented Reynolds-averaged and large eddy simulation models of turbulence”. In: *Physical Review Fluids* 6.5 (2021), p. 050504 (cit. on p. 10).
- [31] Karthik Duraisamy, Gianluca Iaccarino, and Heng Xiao. “Turbulence Modeling in the Age of Data”. In: *Annual Review of Fluid Mechanics* 51.1 (2019), pp. 357–377. URL: <http://dx.doi.org/10.1146/annurev-fluid-010518-040547> (cit. on p. 33).
- [32] E. L. Elte. *IV. Five Dimensional Semiregular Polytope*. Simon & Schuster, 1912 (cit. on p. 128).
- [33] Christer Ericson. *Real-Time Collision Detection*. Elsevier, 2005 (cit. on p. 127).
- [34] Robert Eymard, Thierry Gallouët, and Raphaële Herbin. “Finite volume methods”. In: *Solution of Equation in R HNA (Part 3), Techniques of Scientific Computing (Part 3)*. Vol. 7. Handbook of Numerical Analysis. Elsevier, 2000, pp. 713–1018. URL: <https://www.sciencedirect.com/science/article/pii/S1570865900070058> (cit. on p. 17).
- [35] Joel H. Ferziger and Milovan Perić. *Computational Methods for Fluid Dynamics*. 3rd. Berlin, Heidelberg: Springer, 2002 (cit. on p. 18).
- [36] Robert W. Fox, Alan T. McDonald, and Philip J. Pritchard. *Introduction to Fluid Mechanics*. 8th. John Wiley & Sons, 2015 (cit. on p. 22).
- [37] Elmer G Gilbert, Daniel W Johnson, and S Sathiya Keerthi. “A fast procedure for computing the distance between complex objects in three-dimensional space”. In: *IEEE Journal on Robotics and Automation* 4.2 (1988), pp. 193–203 (cit. on p. 127).
- [38] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2010, pp. 249–256. URL: <https://proceedings.mlr.press/v9/glorot10a.html> (cit. on p. 47).
- [39] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*. Vol. 9. 2010, pp. 249–256. URL: <http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf> (cit. on p. 42).
- [40] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep sparse rectifier neural networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*. JMLR.org. 2011, pp. 315–323 (cit. on p. 35).

- [41] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016 (cit. on p. 36).
- [42] Philip M. Gresho. “The Finite Element Method in Viscous Incompressible Flows”. In: *Recent Advances in Computational Fluid Dynamics*. Ed. by C. C. Chao, S. A. Orszag, and W. Shyy. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 148–190 (cit. on p. 20).
- [43] Hugo Hadwiger. “Minkowskische Addition und Subtraktion beliebiger Punktmengen und die Theoreme von Erhard Schmidt”. In: *Mathematische Zeitschrift* 53.3 (1950), pp. 210–218 (cit. on p. 128).
- [44] Nancy Hall. *Boundary Layer*. 2015 (cit. on p. 22).
- [45] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. 3rd. Morgan Kaufmann, 2011 (cit. on p. 34).
- [46] Douglas M Hawkins. *Identification of Outliers*. Vol. 11. Springer, 1980 (cit. on p. 126).
- [47] Kaiming He et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *arXiv preprint arXiv:1502.01852* (2015). URL: <https://arxiv.org/abs/1502.01852> (cit. on pp. 42, 52).
- [48] Geoffrey Hinton. *Lecture RMSprop: Divide the gradient by a running average of its recent magnitude* (cit. on p. 40).
- [49] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. “A Fast Learning Algorithm for Deep Belief Nets”. In: *Neural Computation* 18 (2006), pp. 1527–1554 (cit. on p. 34).
- [50] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9 (1997), pp. 1735–1780 (cit. on p. 34).
- [51] X. L. D. Huang, X. I. A. Yang, and R. F. Kunz. “Wall-modeled large-eddy simulations of spanwise rotating turbulent channels—Comparing a physics-based approach and a data-based approach”. In: *Physics of Fluids* 31.12 (2019), p. 125105 (cit. on p. 33).
- [52] Peter J. Huber. “Robust Estimation of a Location Parameter”. In: *The Annals of Mathematical Statistics* 35.1 (1964), pp. 73–101. URL: <https://doi.org/10.1214/aoms/1177703732> (cit. on p. 101).
- [53] INRIA Tropics team and INRIA Ecuador team. *TAPENADE: An Automatic Differentiation Engine*. 2021. URL: <https://team.inria.fr/ecuador/en/tapenade/> (cit. on p. 54).
- [54] Xiaodong Jin et al. “Deep Learning for Financial Market Prediction”. In: *Finance Research Letters* 27 (2018), pp. 132–138 (cit. on p. 34).
- [55] W. P. Jones and B. E. Launder. “The Prediction of Laminarization with a Two-Equation Model of Turbulence”. In: *International Journal of Heat and Mass Transfer* 15.2 (1972), pp. 301–314 (cit. on p. 16).
- [56] Georgi Kalitzin et al. “Near-wall behavior of RANS turbulence models and implications for wall functions”. In: *Journal of Computational Physics* 204.1 (2005), pp. 265–291. URL: <https://www.sciencedirect.com/science/article/pii/S0021999104004164> (cit. on pp. 66, 90).

- [57] Soshi Kawai and Johan Larsson. “Wall-modeled large-eddy simulation: Recent applications and guidelines”. In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* 226.5 (2012), pp. 732–742 (cit. on p. 29).
- [58] John Kim, Parviz Moin, and Robert Moser. “Turbulence statistics in fully developed channel flow at low Reynolds number”. In: *Journal of Fluid Mechanics* 177 (1987), pp. 133–166 (cit. on p. 27).
- [59] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. URL: <https://arxiv.org/abs/1412.6980> (cit. on pp. 75, 102).
- [60] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems* 25 (2012), pp. 1097–1105 (cit. on p. 34).
- [61] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet classification with deep convolutional neural networks”. In: *Communications of the ACM* 60.6 (2017), pp. 84–90 (cit. on p. 39).
- [62] Johan Larsson et al. “Large eddy simulation with modeled wall-stress: Recent progress and future directions”. In: *Mechanical Engineering Reviews* 3.1 (2016), pp. 15–25 (cit. on p. 29).
- [63] Zwald Laurent and Lambert-Lacroix Sophie. *The BerHu penalty and the grouped effect*. 2012 (cit. on p. 101).
- [64] Yann LeCun et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1 (1989), pp. 541–551 (cit. on p. 34).
- [65] Yann LeCun et al. “Efficient BackProp”. In: *Neural Networks: Tricks of the Trade* (1998), pp. 9–50. URL: https://doi.org/10.1007/3-540-49430-8_2 (cit. on p. 47).
- [66] James Z. Lee and Katherine L. Garman. “DeepChem: A Deep Learning Framework for Drug Discovery”. In: *Journal of Computational Chemistry* 38 (2017), pp. 1474–1484 (cit. on p. 34).
- [67] Sergey Levine et al. “End-to-End Training of Deep Visuomotor Policies”. In: *Journal of Machine Learning Research* 17 (2016), pp. 1–40 (cit. on p. 34).
- [68] Xiaoyang Li et al. “A Survey on Deep Learning for Cybersecurity”. In: *IEEE Access* 9 (2021), pp. 58028–58050 (cit. on p. 34).
- [69] Ming-Shan Liou. “A Sequel to AUSM: AUSM+”. In: *Journal of Computational Physics* 129.2 (1996), pp. 364–382 (cit. on p. 18).
- [70] Adrián Lozano-Durán and Hyunseut Bae. “Framework for machine-learning-assisted turbulence modeling in large-eddy simulations”. In: *Physical Review Fluids* 6.5 (2021), p. 054602 (cit. on p. 33).
- [71] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. “Rectifier nonlinearities improve neural network acoustic models”. In: *Proceedings of the 30th International Conference on Machine Learning (ICML)*. Vol. 30. 1. PMLR, 2013, pp. 6–11 (cit. on p. 35).
- [72] Ivan Mary et al. *FAST (Flexible Aerodynamic Solver Technology): A compressible flow solver python package*. 2022. URL: <https://w3.onera.fr/FAST/> (cit. on p. 53).

- [73] Warren S. McCulloch and Walter Pitts. “A Logical Calculus of the Ideas Immanent in Nervous Activity”. In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 115–133 (cit. on p. 34).
- [74] Clark B. Millikan. “A critical discussion of turbulent flows in channels and circular tubes”. In: *Proceedings of the Fifth International Congress for Applied Mechanics* (1937), pp. 386–392 (cit. on p. 27).
- [75] Rajat Mittal and Gianluca Iaccarino. “Immersed boundary methods”. In: *Annual Review of Fluid Mechanics* 37 (2005), pp. 239–261 (cit. on pp. 9, 29).
- [76] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012 (cit. on p. 36).
- [77] M. V. Morkovin. “Boundary Layer Transition and Turbulence”. In: *Journal of Fluid Mechanics* 37.1 (1969), pp. 1–37 (cit. on p. 22).
- [78] A. J. Musker. “Explicit expression for the smooth wall velocity distribution in a turbulent boundary layer”. In: *AIAA Journal* 17.6 (1979), pp. 655–657 (cit. on p. 31).
- [79] Vinod Nair and Geoffrey E Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *Proceedings of the 27th International Conference on Machine Learning (ICML)*. USA: Omnipress, 2010, pp. 807–814 (cit. on p. 35).
- [80] B. W. van Oudheusden. “Some Classic Thermal Boundary Layer Concepts Reconsidered (and their Relation to Compressible Couette Flow)”. In: *IUTAM Symposium on One Hundred Years of Boundary Layer Research*. Ed. by G. E. A. Meier, K. R. Sreenivasan, and H.-J. Heinemann. Dordrecht: Springer Netherlands, 2006, pp. 425–434 (cit. on p. 65).
- [81] Miltiadis V. Papalexandris. “On the applicability of Stokes’ hypothesis to low-Mach-number flows”. In: *Continuum Mechanics and Thermodynamics* 32 (2020), pp. 1245–1249. URL: <https://doi.org/10.1007/s00161-019-00785-z> (cit. on p. 15).
- [82] Romain Paris. “Potential and challenges of reinforcement learning for flow control”. English. Fluid mechanics [physics.class-ph]. Institut Polytechnique de Paris, 2022. URL: <https://tel.archives-ouvertes.fr/tel-04117830v2> (cit. on pp. 33, 40).
- [83] George I. Park and Parviz Moin. “An improved dynamic non-equilibrium wall-model for large eddy simulation”. In: *Physics of Fluids* 28.4 (2016), p. 045103 (cit. on p. 30).
- [84] YeongHyeon Park. “Concise Logarithmic Loss Function for Robust Training of Anomaly Detection Model”. In: (2022). URL: <https://arxiv.org/abs/2201.05748> (cit. on p. 72).
- [85] Ugo Piomelli. “Wall-layer models for large-eddy simulations”. In: *Progress in Aerospace Sciences* 44.6 (2008), pp. 437–446 (cit. on p. 9).
- [86] Ugo Piomelli and Elias Balaras. “Wall-layer models for large-eddy simulations”. In: *Annual Review of Fluid Mechanics* 34.1 (2002), pp. 349–374 (cit. on p. 29).
- [87] Stephen B. Pope. *Turbulent Flows*. Cambridge University Press, 2000 (cit. on pp. 9, 14, 25, 27).
- [88] Ludwig Prandtl. “Über Flüssigkeitsbewegung bei sehr kleiner Reibung”. In: *Verhandlungen des III. Internationalen Mathematiker-Kongresses*. Heidelberg, Germany, 1904, pp. 484–491 (cit. on p. 21).

- [89] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. “On the Convergence of Adam and Beyond”. In: *Proceedings of the 6th International Conference on Learning Representations (ICLR 2018)*. 2018. URL: <https://arxiv.org/abs/1904.09237> (cit. on p. 40).
- [90] Osborne Reynolds. “An Experimental Investigation of the Circumstances which Determine Whether the Motion of Water Shall be Direct or Sinuous, and of the Law of Resistance in Parallel Channels”. In: *Philosophical Transactions of the Royal Society of London* 174 (1883), pp. 935–982. URL: <https://archive.org/details/philtrans02197454> (cit. on p. 22).
- [91] P.L. Roe. “Approximate Riemann solvers, parameter vectors, and difference schemes”. In: *Journal of Computational Physics* 43.2 (1981), pp. 357–372. URL: <https://www.sciencedirect.com/science/article/pii/0021999181901285> (cit. on pp. 18, 53).
- [92] M. Romanelli et al. “Data-driven wall models for Reynolds Averaged Navier–Stokes simulations”. In: *International Journal of Heat and Fluid Flow* Volume 99 (2023), p. 109097 (cit. on pp. 11, 64, 90, 95, 97, 98, 99, 110, 117, 119, 121, 122).
- [93] Frank Rosenblatt. “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain”. In: *Psychological Review* 65 (1958), pp. 386–408 (cit. on p. 34).
- [94] C. Rumsey et al. “Recent Updates to the CFD General Notation System (CGNS)”. In: *50th AIAA Aerospace Sciences Meeting*. Nashville, TN: American Institute of Aeronautics and Astronautics, 2012 (cit. on p. 53).
- [95] Christopher Rumsey. *2D Bump-in-channel Verification Case*. 2021. URL: <https://turbmodels.larc.nasa.gov/bump.html> (cit. on p. 56).
- [96] Christopher L. Rumsey, Dan H. Neuhart, and Michael A. Kegerise. “The NASA Juncture Flow Experiment: Goals, Progress, and Preliminary Testing”. In: *AIAA Scitech 2019 Forum*. American Institute of Aeronautics and Astronautics (AIAA). 2019 (cit. on p. 33).
- [97] Hermann Schlichting and Klaus Gersten. *Boundary-Layer Theory*. 8th ed. Berlin, Heidelberg: Springer, 2000 (cit. on pp. 21, 22, 27).
- [98] M. L. Shur et al. “Turbulence Modeling in Rotating and Curved Channels: Assessing the Spalart-Shur Correction”. In: *AIAA Journal* 38.5 (2000), pp. 784–792 (cit. on p. 16).
- [99] B.W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall/CRC, 1986 (cit. on p. 51).
- [100] Arthur M.O. Smith and Norman Gamberoni. “Laminar Boundary Layer Oscillations and Transition on a Flat Plate”. In: *Journal of the Aeronautical Sciences* 20.11 (1953), pp. 793–800 (cit. on p. 23).
- [101] Jan Snyman. *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*. Springer Science & Business Media, 2005 (cit. on p. 39).
- [102] P. Spalart and S. Allmaras. “A one-equation turbulence model for aerodynamic flows”. In: *30th Aerospace Sciences Meeting and Exhibit*. URL: <https://arc.aiaa.org/doi/abs/10.2514/6.1992-439> (cit. on p. 16).

- [103] P. R. Spalart and A. V. Garbaruk. “Correction to the Spalart-Allmaras Turbulence Model, Providing More Accurate Skin Friction”. In: *AIAA Journal* 58.5 (2020), pp. 1903–1905 (cit. on p. 16).
- [104] P. R. Spalart et al. “Comments on the feasibility of LES for wings and on a hybrid RANS/LES approach”. In: *Proceedings of the First AFOSR International Conference on DNS/LES*. 1997, pp. 137–147 (cit. on p. 30).
- [105] Philippe R Spalart. “Detached-eddy simulation”. In: *Annual Review of Fluid Mechanics* 41 (2009), pp. 181–202 (cit. on p. 30).
- [106] D.B. Spalding. “A single formula for the law of the wall”. In: *Journal of Applied Mechanics* 28.3 (1961), pp. 455–458 (cit. on p. 31).
- [107] Michael Steininger et al. “Density-based weighting for imbalanced regression”. In: *Machine Learning* 110 (2021), pp. 2187–2211 (cit. on pp. 51, 101).
- [108] George Stokes. “On the Effect of the Internal Friction of Fluids on the Motion of Pendulums”. In: *Transactions of the Cambridge Philosophical Society* 9 (1851), pp. 8–106 (cit. on p. 22).
- [109] Ilya Sutskever et al. “On the importance of initialization and momentum in deep learning”. In: *Proceedings of the 30th international conference on machine learning (ICML-13)*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Atlanta, GA, 2013, pp. 1139–1147. URL: <http://proceedings.mlr.press/v28/sutskever13.pdf> (cit. on p. 40).
- [110] Yoshiharu Tamaki, Motoshi Harada, and Taro Imamura. “Near-Wall Modification of Spalart–Allmaras Turbulence Model for Immersed Boundary Method”. In: *AIAA Journal* 55.9 (2017), pp. 3027–3039. URL: <https://doi.org/10.2514/1.J055824> (cit. on pp. 31, 106, 115, 120).
- [111] Bo Tang and Haibo He. “A local density-based approach for outlier detection”. In: *Neurocomputing* 241 (2017), pp. 171–180. URL: <https://www.sciencedirect.com/science/article/pii/S0925231217303302> (cit. on p. 126).
- [112] L. Temmerman and M.A. Leschziner. “Large eddy simulation of separated flow in a streamwise periodic channel constriction: Influence of grid resolution and subgrid model”. In: *International Journal of Heat and Fluid Flow* 24.2 (2003), pp. 157–180 (cit. on p. 29).
- [113] Igor V. Tetko, David J. Livingstone, and Alexander I. Luik. “Neural network studies. 1. Comparison of Overfitting and Overtraining”. In: *Journal of Chemical Information and Modeling* 35.5 (1995), pp. 826–833. URL: <https://pubs.acs.org/doi/pdf/10.1021/ci00027a006> (cit. on p. 42).
- [114] *The Official CGNS Home Page*. URL: <https://cgns.github.io/cgns-modern.github.io/index.html> (cit. on p. 53).
- [115] Eleuterio F. Toro, Michael Spruce, and W. Speares. “Restoration of the contact surface in the HLL-Riemann solver”. In: *Shock Waves* 4.1 (1994), pp. 25–34 (cit. on p. 18).
- [116] Gino Van Den Bergen. “A fast and robust GJK implementation for collision detection of convex objects”. In: *Journal of Graphics Tools* 4.2 (1999), pp. 7–25 (cit. on p. 127).

- [117] Bram van Leer. “Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov’s method”. In: *Journal of Computational Physics* 32.1 (1979), pp. 101–136. URL: <https://www.sciencedirect.com/science/article/pii/0021999179901451> (cit. on p. 19).
- [118] Ashish Vaswani et al. “Attention is All You Need”. In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017, pp. 5998–6008 (cit. on p. 34).
- [119] H. K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. 2nd. Harlow: Pearson, 2007 (cit. on p. 18).
- [120] Pedro S. Volpiani et al. “Data-driven compressibility transformation for turbulent wall layers”. In: *Physical Review Fluids* 5.5 (2020), 052602(R) (cit. on p. 33).
- [121] Shumin Wang and Zhaohong Deng. “Stratified sampling for feature subspace selection in random forests for high-dimensional data”. In: *Pattern Recognition* 45.9 (2012), pp. 3239–3247 (cit. on p. 50).
- [122] H. Werner and H. Wengle. “Large-Eddy Simulation of Turbulent Flow Over and Around a Cube in a Plate Channel”. In: *Turbulent Shear Flows 8*. Ed. by Franz Durst et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 155–168 (cit. on p. 67).
- [123] Frank M. White. *Viscous Fluid Flow*. 3rd. McGraw-Hill Education, 2006 (cit. on p. 22).
- [124] David C. Wilcox. “Reassessment of the scale-determining equation for advanced turbulence models”. In: *AIAA Journal* 26.11 (1988), pp. 1299–1310. URL: <https://doi.org/10.2514/3.10041> (cit. on p. 16).
- [125] Saining Xie and Zhuowen Tu. “Holistically-nested edge detection”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1395–1403 (cit. on p. 50).
- [126] X. I. A. Yang et al. “Integral wall model for large eddy simulations of wall-bounded turbulent flows”. In: *Physics of Fluids* 27.2 (2015), p. 025112 (cit. on p. 32).
- [127] X. I. A. Yang et al. “Predictive large-eddy-simulation wall modeling via physics-informed neural networks”. In: *Physical Review Fluids* 4 (2019), p. 034602 (cit. on p. 33).
- [128] Xiang I A Yang et al. “Integral wall model for large eddy simulations of wall-bounded flows”. In: *Physics of Fluids* 29.9 (2017), p. 091701 (cit. on p. 29).
- [129] In-Kwon Yeo and Richard A. Johnson. “A new family of power transformations to improve normality or symmetry”. In: *Biometrika* 87.4 (2000), pp. 954–959. URL: <https://doi.org/10.1093/biomet/87.4.954> (cit. on pp. 48, 100).
- [130] D. Zhou et al. “Multi-agent reinforcement learning for wall modeling in LES of flow over periodic hills”. In: *Physical Review Fluids* 7.2 (2022), p. 024604 (cit. on p. 33).
- [131] Zhideng Zhou, Guowei He, and Xiaolei Yang. “Wall model based on neural networks for les of turbulent flows over periodic hills”. In: *Physical Review Fluids* 6.5 (2021), pp. 1–30 (cit. on pp. 33, 50, 64, 72).
- [132] Zhideng Zhou et al. “A wall model learned from the periodic hill data and the law of the wall”. In: *Physics of Fluids* 35.5, 055108 (2023), p. 055108 (cit. on p. 33).

Lois de paroi à apprentissage profond pour simulations aérodynamiques

Résumé : Les simulations aux équations de Navier-Stokes moyennées (RANS) sont largement utilisées dans le domaine industriel. Cependant, leur précision dépend fortement de la solution des écoulements en proche paroi, nécessitant typiquement un maillage très fin pour capturer correctement les forts gradients se développant dans la couche limite. Cela entraîne un impact considérable sur le coût de calcul de la simulation. Les lois de paroi permettent d'alléger ces calculs en remplaçant la résolution coûteuse de la couche limite par une modélisation. Dans ce contexte, des approches basées sur l'apprentissage profond sont explorées et la flexibilité des réseaux de neurones offre un potentiel indéniable dans la modélisation des écoulements pariétaux. Ce travail vise à développer une loi de paroi basée sur l'apprentissage profond qui peut reproduire avec précision l'évolution de la région interne de la couche limite, fournissant ainsi des conditions aux limites valables pour les calculs RANS se déroulant loin de la paroi. Une approche préliminaire consiste à entraîner un réseau de neurones sur des données résolues jusqu'à la paroi pour reconstruire des profils de vitesse adimensionnelle et modéliser l'évolution de la couche limite. Conformément aux lois de paroi analytiques, la vitesse est fonction de la distance à la paroi et du gradient de pression. Ces variables sont adimensionnalisées à l'aide d'une vitesse de frottement caractéristique, qui est estimée de manière itérative à l'aide d'un algorithme de Newton-Raphson. Pour réduire le coût associé à l'estimation itérative de la contrainte visqueuse à la paroi, une nouvelle approche entièrement basée sur l'apprentissage profond a été développée. Elle impose directement le frottement à l'interface entre la région modélisée et le calcul RANS, fixant la dérivée normale du champ de vitesse, qui est estimée à l'aide de deux réseaux de neurones interconnectés : l'un estimant la contrainte de cisaillement à la paroi et l'autre évaluant la dérivée adimensionnelle de la vitesse. Les réseaux de neurones sont entraînés sur des simulations RANS entièrement résolues d'écoulements turbulents sur diverses géométries de bosses bidimensionnelles. Les performances, en termes de précision et coût computationnel, de ce modèle sont ensuite comparées à des calculs résolus jusqu'à la paroi pour des configurations d'écoulements non incluses dans le jeu de données d'entraînement.

Mots-clés : Dynamique des fluides numérique, simulations RANS, Loi de paroi, Apprentissage profond

Deep Wall Models for Aerodynamic Simulations

Abstract: Reynolds-Averaged Navier-Stokes (RANS) simulations are widely used in the industrial domain. However, their accuracy heavily relies on the solution of near-wall flows, typically requiring a very fine mesh to properly capture the steep gradients developing in the boundary layer. This results in a substantial impact on the computational cost of the simulation. Wall laws allow to speed up of these calculations by replacing the costly resolution of the boundary layer with modeling. In this context, deep learning-based approaches are being explored and the flexibility of neural networks offers undeniable potential in modeling near-wall flows. This work aims to develop a wall law based on deep learning that can accurately reproduce the evolution of the internal region of the boundary layer, thereby providing boundary conditions for the RANS calculations occurring far from the wall. A preliminary approach involves training a neural network on wall-resolved data to reconstruct dimensionless velocity profiles and model the evolution of the boundary layer near the wall. In accordance with analytical wall laws, velocity is a function of the distance to the wall and the pressure gradient, with these variables being non-dimensionalized using a characteristic friction velocity, which is iteratively estimated using a Newton-Raphson algorithm. To reduce the cost associated with the iterative estimation of the wall shear stress, a new approach entirely based on deep learning has been developed. It directly imposes the friction at the interface between the modeled region and the RANS calculation, fixing the normal derivative of the velocity field, which is estimated using two interconnected neural networks: one estimating the wall shear stress and the other evaluating the dimensionless normal derivative of the velocity. The neural networks are trained on fully resolved RANS simulations of turbulent flows over various two-dimensional bump geometries. The performance of this wall model, in term of accuracy and computational cost, is then compared to wall-resolved calculations for flow configurations not included in the training dataset.

Keywords: Computational fluid dynamics, RANS simulations, Wall model, Deep learning
