Politecnico di Torino

Politecnico di Torino

Master's Degree in Aerospace Engineering
A.A. 2025/2026
Graduation Session: March 2026

Graph Neural Networks for Surrogate Modeling of Fluid Flows

Supervisors:

Prof. Andrea Ferrero
Ing. Leonardo Stumpo

Candidate:

Carmelo Alia

Acknowledgments

I would like to express my sincere gratitude to all those who have supported me throughout the development of this thesis. First and foremost, I would like to thank my supervisors, Prof. Andrea Ferrero and Ing. Leonardo Stumpo, for their guidance, valuable insights, and constant availability. Their suggestions and constructive feedback have been fundamental to the completion of this work. I am deeply grateful to my parents and my sister for their unconditional support and encouragement. Their trust and motivation have always pushed me to strive for excellence and have allowed me to reach this important milestone. I would also like to thank my closest friends, who have remained present and supportive despite the distance, as well as my colleagues, who have made this academic journey both stimulating and enjoyable. Finally, I would like to thank my relatives for their continuous encouragement, and my girlfriend Irene for her strength, patience, and constant support. Thank you for always being by my side.

Abstract

This thesis investigates the use of Graph Neural Networks (GNNs) as surrogate models for computational fluid dynamics problems characterized by fundamentally different physical and mathematical properties. The objective is to evaluate whether the same message-passing architecture, trained separately on different datasets, can effectively model fluid flows governed by distinct physical assumptions and mathematical properties.

In the first case a convergent–divergent nozzle governed by the 2D compressible Euler equations is considered. A parametric dataset is generated by varying the chamber-to-exit pressure ratio, leading to different flow regimes ranging from fully subsonic to mixed elliptic–hyperbolic behavior with shock formation. The GNN is trained to perform node-wise regression of Mach number and static pressure fields. While the model is trained to approximate the steady-state solution, its intended role is to provide physically consistent initial conditions that can accelerate the convergence of conventional time-marching solvers.

In the second case a viscous incompressible flow over a backward-facing step is simulated in ANSYS by varying the Reynolds number. Here, the same architecture is employed to predict the streamwise velocity component and pressure distribution, with the aim of accurately reproducing separation phenomena, wall shear stress and viscous boundary-layer effects.

Both models are trained using a mean squared error loss and evaluated through qualitative field comparisons and normalized relative error metrics. The results demonstrate that the same Graph Neural Network architecture, although trained independently for each case, can effectively capture distinct fluid dynamic behaviors arising from different governing equations, highlighting the flexibility of GNNs for surrogate modeling in fluid dynamics applications.

Contents

1 Introduction **1**
- 1.1 Motivation and Context . . . 1
- 1.2 Machine Learning for CFD . . . 2
- 1.3 Research Objective . . . 4
- 1.4 Thesis Structure . . . 5

2 Graph Neural Network **9**
- 2.1 Neural Network . . . 9
- 2.2 Graphs . . . 23
- 2.3 Graph Neural Network . . . 30
- 2.4 Neural Operator . . . 36

3 The Governing Equations of Fluid Dynamics **43**
- 3.1 Conservation of Mass . . . 45
- 3.2 Conservation of Momentum . . . 46
- 3.3 Conservation of Energy . . . 48
- 3.4 Closure of the governing equation . . . 49
- 3.5 From Navier–Stokes to Euler equations . . . 51

4 Parametric Prediction of Nozzle Flow **55**
- 4.1 Dataset Generation . . . 57
  - 4.1.1 Geometry and Mesh Generation . . . 57
  - 4.1.2 Numerical Discretisation . . . 60
  - 4.1.3 Parametric CFD Simulations and Data Pre – Procssing 62
- 4.2 Graph-based representation and Message – Passing architecture 63
  - 4.2.1 Graph construction . . . 63
  - 4.2.2 Design of the Message Passing Architecture . . . 64
- 4.3 Training and Validation Procedure . . . 67
- 4.4 Results . . . 68
- 4.5 Discussion of Results . . . 86

5 Parametric Prediction of Backward-Facing Step Flow 89
5.1 Dataset Generation 91
5.1.1 Geometry and Mesh Generation 91
5.1.2 Numerical Discretisation 94
5.1.3 Parametric CFD Simulations and Data Pre – Procssing 95
5.2 Graph-based representation and Message – Passing architecture 97
5.2.1 Graph construction 97
5.2.2 Design of the Message Passing Architecture 98
5.3 Training Procedure 100
5.4 Results 101
5.5 Discussion of Results 114

6 Conclusions 119

A Pressure-field comparisons for the MSE-only model 123

B Pressure-field comparisons for the blended loss model 127

Chapter 1

Introduction

1.1 Motivation and Context

Computational Fluid Dynamics (CFD) has become one of the most important tools for the analysis, prediction, and design of engineering systems involving fluid flows [4]. Over the past decades, the rapid growth in computational power and numerical methods has enabled the simulation of increasingly complex physical phenomena, ranging from aerospace propulsion systems to turbomachinery, environmental flows, and industrial fluid transport processes. By solving the governing conservation equations of mass, momentum, and energy, CFD allows engineers and researchers to investigate flow structures, pressure distributions, shock waves, separation phenomena, and boundary-layer development with a level of detail that would be difficult, expensive, or even impossible to obtain experimentally. Despite its versatility and predictive capability, high-fidelity CFD simulations remain computationally demanding. The numerical solution of nonlinear partial differential equations, such as the Euler or Navier–Stokes equations, requires spatial discretization over fine computational meshes and iterative solution procedures that may involve thousands of time steps or nonlinear iterations. The computational burden increases significantly when dealing with parametric studies, design optimization or real-time control applications, where the governing equations must be solved repeatedly for different boundary conditions or geometric configurations. In such scenarios, the repeated evaluation of complex numerical solvers often represents the primary bottleneck in the engineering workflow. Moreover, as mesh resolution increases to capture localized phenomena such as shock waves or near-wall viscous effects, the computational complexity increases significantly, resulting in higher memory requirements and longer convergence times. Even with modern high-performance com-

puting infrastructures, large-scale simulations may remain computationally demanding, particularly when fine spatial and temporal resolutions are required. This limitation becomes particularly critical in industrial environments, where rapid design iterations and decision-making processes are essential. For these reasons, the development of reduced-order models and surrogate approaches has become an active area of research within the CFD community. Instead of solving the governing equations from scratch for every new configuration, surrogate models aim to approximate the mapping between input parameters (e.g., boundary conditions, Reynolds number, pressure ratios) and the resulting flow fields. If sufficiently accurate, such models can provide substantial speed-ups while preserving the essential physical characteristics of the solution. The challenge, however, lies in designing surrogate models that are both computationally efficient and physically consistent, especially when the underlying governing equations exhibit complex nonlinear behavior.

1.2 Machine Learning for CFD

In recent years, machine learning techniques have increasingly been explored as complementary tools to traditional numerical solvers. Rather than explicitly discretizing and solving the governing partial differential equations at each iteration, data-driven models aim to learn the functional relationship between input parameters and solution fields directly from data. In the context of fluid dynamics, this typically involves approximating a nonlinear mapping of the form:

$$\mathcal{F} : (\text{geometry}, \text{boundary conditions}, \text{physical parameters}) \longrightarrow \text{flow field}$$

If successfully learned, such a mapping can be evaluated at a fraction of the computational cost required by classical CFD solvers. This potential computational acceleration makes machine learning particularly attractive for applications involving parametric studies, optimization procedures or real-time control, where repeated high-fidelity simulations would otherwise be required. Early applications of deep learning in fluid mechanics largely relied on convolutional neural networks (CNNs), which are particularly effective when the computational domain is discretized on structured Cartesian grids [2]. CNNs exploit translational invariance and local receptive fields, making them well suited for image-like representations of flow variables. However, most industrial CFD simulations employ unstructured meshes to accurately capture complex geometries and localized physical phenomena. In such cases, the

regular grid assumption underlying convolutional architectures is no longer valid, and projecting solution fields onto structured tensors may introduce interpolation errors or geometric distortions. To overcome these limitations, Graph Neural Networks (GNNs) have emerged as a natural framework for learning on mesh-based data. In a graph representation, mesh vertices are treated as nodes, while edges encode the topological connectivity between neighboring elements. This formulation allows neural networks to operate directly on unstructured discretizations without imposing artificial regularity. The core mechanism underlying most GNN architectures is the message-passing paradigm, introduced in a general form by Gilmer et al. [1]. In this framework, node features are iteratively updated by aggregating information from neighboring nodes through learnable transformations. Through successive layers, information propagates across the computational domain, enabling the network to capture both local interactions and larger-scale flow structures. From a physical standpoint, this iterative exchange of information can be interpreted as a learned representation of the local coupling mechanisms that arise from the discretization of conservation laws. In compressible flows, such couplings are associated with nonlinear flux interactions and wave propagation, whereas in incompressible viscous flows they are strongly influenced by pressure coupling and diffusive effects. By stacking multiple message-passing layers, the network progressively enlarges its effective receptive field, allowing non-local dependencies to emerge without explicitly solving the governing equations. Another important advantage of GNNs lies in their permutation invariance and their ability to naturally handle irregular connectivity patterns. This makes them particularly suitable for CFD applications, where mesh topology and node distribution may vary between simulations. Recent work by Pfaff et al. [3] demonstrated that graph-based architectures can successfully learn mesh-based physical simulations, highlighting their potential for modeling complex dynamical systems governed by partial differential equations. Despite these promising developments, to the best of our knowledge, most existing studies focus on a single physical regime or a specific class of governing equations. The question of whether a single message-passing architecture can effectively approximate solutions arising from fundamentally different mathematical structures such as hyperbolic compressible flows and elliptic dominated incompressible flows remains largely unexplored. Investigating this structural flexibility constitutes one of the central motivations of the present thesis.

1.3 Research Objective

The primary goal of this thesis is to investigate the structural flexibility of Graph Neural Networks when applied to fluid dynamic problems characterized by fundamentally different physical and mathematical properties. Rather than designing case-specific neural architectures, the same message-passing framework is adopted in two distinct CFD scenarios and trained independently on their respective datasets. The first configuration involves a compressible flow in a convergent–divergent nozzle governed by the two-dimensional Euler equations. This problem is characterized by nonlinear wave propagation, mixed elliptic–hyperbolic behavior, and the possible formation of shock waves. The second configuration concerns a viscous incompressible flow over a backward-facing step, governed by the Navier–Stokes equations and dominated by diffusive effects, pressure coupling, boundary-layer development, and flow separation. From a mathematical viewpoint, these two systems exhibit markedly different properties. Compressible inviscid flows are primarily hyperbolic in nature and may develop discontinuities, whereas incompressible viscous flows involve elliptic constraints and diffusive mechanisms that introduce fundamentally different coupling structures. Consequently, the corresponding numerical solutions present distinct spatial behaviors and interaction patterns. The central research question addressed in this work is therefore the following: can a single message-passing Graph Neural Network architecture, without structural modifications, effectively approximate solution fields arising from governing equations with such different mathematical characteristics?

In both case studies, the network is trained to perform node-wise regression of physically relevant quantities. However, the intended purpose of the model differs between the two case studies. In the compressible nozzle configuration, the objective is not to replace conventional CFD solvers, but rather to evaluate whether the learned surrogate can generate physically consistent flow fields that may serve as high-quality initial conditions, potentially accelerating the convergence of traditional time-marching numerical schemes. In contrast, for the backward-facing step case, the goal is more ambitious: the model is trained to directly approximate the steady-state solution fields, thereby assessing its capability to act as a direct surrogate for the CFD solution within the considered parametric range. By systematically evaluating the performance of the same architecture across these heterogeneous regimes, this thesis aims to provide insight into the generalization capabilities and limitations of Graph Neural Networks in computational fluid dynamics applications.

1.4 Thesis Structure

The remainder of this thesis is organized as follows:

1. **Chapter 2: Graph Neural Network.** This chapter introduces the theoretical foundations of Graph Neural Networks. After a general overview of neural network models, the graph representation of data is discussed, followed by a detailed description of graph-based architectures and message-passing mechanisms. The chapter concludes with a brief discussion on neural operators.

2. **Chapter 3: The Governing Equations of Fluid Dynamics.** This chapter presents the conservation equations of mass, momentum, and energy. Particular emphasis is placed on the compressible Euler equations and their relation to the Navier–Stokes system.

3. **Chapter 4: Parametric Prediction of Nozzle Flow.** This chapter describes the first case study, focusing on compressible flow in a convergent–divergent nozzle. The dataset generation, graph construction, network architecture, training procedure, and results are presented.

4. **Chapter 5: Parametric Prediction of Backward-Facing Step Flow.** This chapter addresses the second case study, involving viscous incompressible flow over a backward-facing step at varying Reynolds numbers. The dataset construction, graph modeling strategy, training procedure, and results are discussed.

Bibliography

[1] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

[2] Xiaowei Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD Workshop on Machine Learning in Physical Sciences*, 2016.

[3] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations (ICLR)*, 2021.

[4] H. K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson, 2 edition, 2007.

Chapter 2

Graph Neural Network

The aim of this chapter is, in the first part, to introduce the concept of a neural network (**Neural Network**), illustrating its operating principles and the reasons behind its widespread success. A comparison between the main differences in the use of a **Shallow Neural Network** and a **Deep Neural Network** will also be provided. Subsequently, some mathematical structures known as **graphs** will be introduced, followed by an analysis of how neural networks can be integrated with graph structures, leading to the models known as **Graph Neural Networks** (GNNs). The main advantages of GNNs over conventional architectures such as **Convolutional Neural Networks** (CNNs) will also be highlighted. The chapter concludes with a brief introduction to **Neural Operators**, a class of deep learning models designed to learn operators between functional spaces. Particular attention will be devoted to the **Graph Neural Operator** (GNO), which combines graph-based representations with the **operator learning** framework. It should be emphasized that the purpose of this chapter is to provide the reader with an introductory overview of the topics discussed, aimed at supporting the understanding of the applications presented in the subsequent chapters. Therefore, it does not represent an exhaustive treatment of the subjects addressed.

2.1 Neural Network

A neural network can be interpreted as a function

$$\mathbf{y} = \mathbf{f}[\mathbf{x}; \mathbf{w}, \mathbf{b}],$$

which, given an input $\mathbf{x} \in \mathbb{R}^m$, returns an output $\mathbf{y} \in \mathbb{R}^n$. To understand how a neural network represents a function and the role played by the parameters

w and **b**, it is useful to first describe its architecture. As shown in Figure 2.1, every neural network is composed of three main components:

- **Input layer**: the initial layer, where the input values of the model are provided.
- **Hidden layer(s)**: the central part of the network, consisting of one or more intermediate layers made up of computational units called **neurons** or **hidden units**. These layers are referred to as "hidden" because they are not directly visible to the user and do not produce the final output of the model.
- **Output layer**: the final layer of the network, responsible for producing the output values.

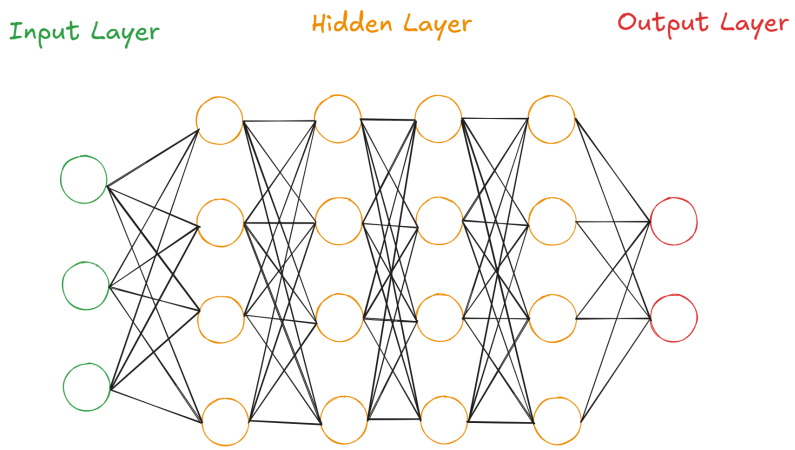


Figure 2.1: Structure of a neural network. Image created by the author.

Once the overall structure of a neural network has been introduced, we can proceed by analyzing the simplest architecture that can be constructed, known as a **Shallow Neural Network**.

Shallow Neural Network

A *shallow neural network* takes this name because it is a neural network composed of a single hidden layer, as illustrated in Figure 2.2. The fundamental idea underlying neural networks is that the value of each neuron in a given layer depends, in the case of a fully connected network, on all the values of the neurons in the previous layer. In the case of the network under consideration:

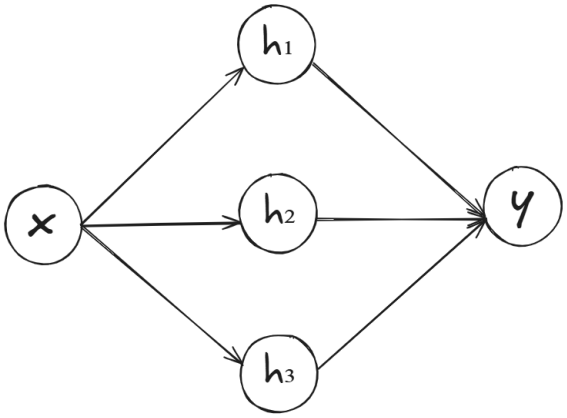


Figure 2.2: Structure of a shallow network. Image created by the author.

$$h_1^{[1]} = a(b_{11}^{[1]} + w_{11}^{[1]}x) \tag{2.1}$$
$$h_2^{[1]} = a(b_{21}^{[1]} + w_{21}^{[1]}x) \tag{2.2}$$
$$h_3^{[1]} = a(b_{31}^{[1]} + w_{31}^{[1]}x) \tag{2.3}$$

where $\vec{w}^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{21}^{[1]} & w_{31}^{[1]} \end{bmatrix}$ and $\vec{b}^{[1]} = \begin{bmatrix} b_{11}^{[1]} & b_{21}^{[1]} & b_{31}^{[1]} \end{bmatrix}$ represent respectively the **weights** and the **biases** associated with the first layer. Regarding the adopted notation, the superscript, whether referring to the neuron values (h) or to the parameters (w, b), always indicates the layer of the network being considered. The subscript, on the other hand, assumes a different meaning depending on the context: in the case of neurons, it identifies a specific neuron belonging to the layer, whereas in the case of the parameters, the first index indicates the receiving neuron and the second the originating neuron, thus specifying the connection to which the parameter is associated. The function $a[.]$ represents a nonlinear function known as the **activation function**. There exist several types of activation functions, and among the most commonly used in the literature we find:

- **Logistic Sigmoid Function.** (Figure 2.3):

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

- **Rectified Linear Unit.** (Figure 2.4):

$$\text{ReLU}(x) = \max(0, x) \tag{2.5}$$

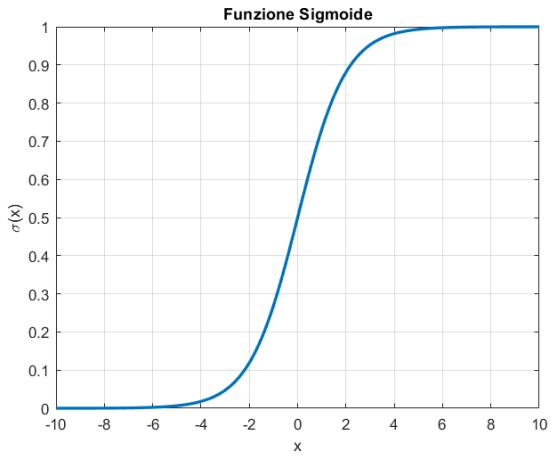


Figure 2.3: Logistic sigmoid function. Image created by the author.

Deep Neural Network

A *Deep Neural Network*, unlike a *Shallow Neural Network*, takes this name because it is a neural network composed of more than one hidden layer, as illustrated in Figure 2.1. The general operating principle remains analogous to that of a shallow neural network; what primarily changes is the notation, which becomes more articulated as the depth of the network increases. In compact form, the values of the neurons belonging to a given layer $[L]$ can be expressed as:

$$\vec{h}^{[L]} = a\left(\mathbf{W}^{[L]} \cdot \vec{h}^{[L-1]} + \vec{b}^{[L]}\right) \tag{2.6}$$

where $\mathbf{W}^{[L]}$ is a matrix in which each row contains the weights connecting all the neurons of the previous layer $[L-1]$ to the neuron in layer $[L]$ corresponding to that same row. In order to understand why neural networks are today among the most widely used architectures compared to other machine learning models, it is necessary to introduce one of their most relevant properties, namely their ability to work as **universal function approximators**. This property is formally established in the so-called **Universal Approximation Theorem**, whose statement is the following:

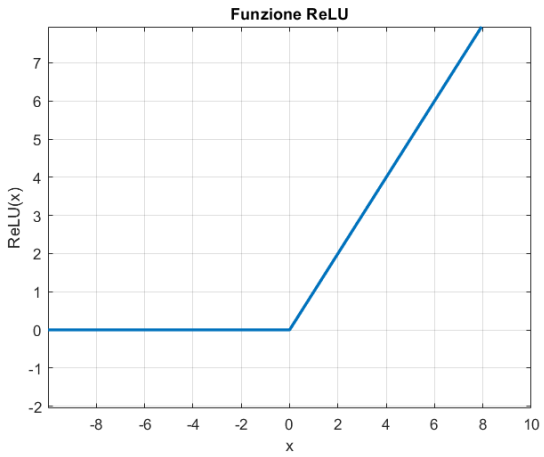


Figure 2.4: Rectified Linear Unit. Image created by the author.

Universal Approximation Theorem

Let $C(K)$ be the space of continuous functions defined on a compact set $K \subset \mathbb{R}^n$. Then, for every continuous function $f \in C(K)$ and for every $\varepsilon > 0$, there exists a feedforward neural network $\hat{f}$ with a single hidden layer and a finite number of neurons such that:

$$|f(x) - \hat{f}(x)| < \varepsilon \quad \text{for every } x \in K.$$

Although this theorem is stated for a Shallow Neural Network, its validity can also be extended to a Deep Neural Network. The meaning of the theorem defined above implies that a neural network $\hat{f}$ is able to approximate a function $f(x)$ with an arbitrarily small accuracy level ε, provided that a sufficient number of neurons is used in the hidden layer. This theorem therefore provides a mathematical interpretation of why neural networks are capable of solving complex problems in various fields such as image recognition and natural language processing. A fundamental assumption of this theorem is that the activation function employed must be a continuous, bounded, non-constant and, most importantly, **nonlinear** function on the compact set $K \subset \mathbb{R}^n$ under consideration. It is precisely the nonlinearity of the activation function that plays a crucial role in enabling the Neural Network to model complex problems. It should also be emphasized that, although this theorem guarantees the existence of a neural network capable of approximating functions, it does not provide any practical guidance on how to construct such a network or how to train it efficiently. In this paragraph, a

mathematical proof of the theorem will not be provided; however, Figure 2.5 allows its meaning to be understood qualitatively. It can be observed that, as the number of neurons in a *shallow neural network* with ReLU activation increases, the approximating function $\hat{f}(x)$ exhibits a growing number of linear regions. Consequently, the function $\hat{f}(x)$, represented by the solid line, approximates the target function $f(x)$, indicated by the dashed line, with increasing accuracy.

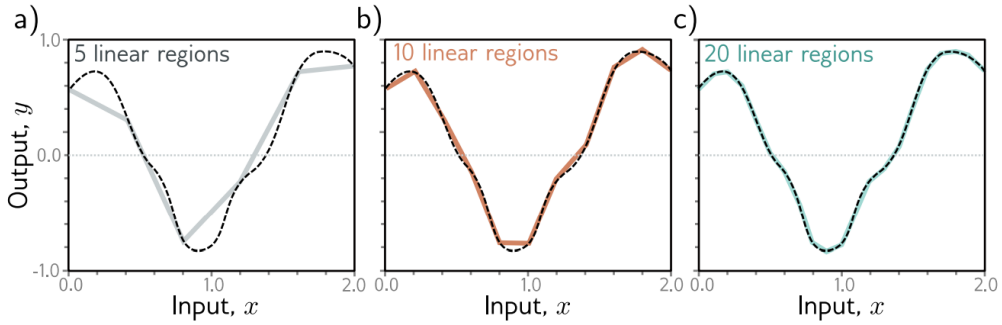


Figure 2.5: Qualitative illustration of the Universal Approximation Theorem. Image taken from Prince [5].

Shallow vs. Deep Neural Network

In the previous paragraph, it was shown that both *Shallow Neural Networks* and *Deep Neural Networks* possess the ability to approximate arbitrary functions, thus belonging to the class of universal approximators. At this point, it is natural to ask under which circumstances it is preferable to use a shallow network rather than a deep one. The aim of this subsection is to illustrate the reasons why, in recent literature, *Deep Neural Networks* are generally preferred over *Shallow Neural Networks*. The main reason can be visualized in Figure 2.6.
In this figure, several neural network architectures are considered, all employing the ReLU function as the activation function. Depending on how we move along the graph, it is possible to vary either the number of neurons in each layer or the number of layers itself, indicated by the letter K. From the figure, it can be observed that, for the same number of linear regions generated by the output function, a *deep neural network* with five layers requires a significantly smaller number of parameters compared to a shallow network with a single layer. This implies that, for the same expressive capacity (in terms of linear regions), a deep network is computationally less expensive than a shallow one. The figure can also be interpreted in the opposite way: for the

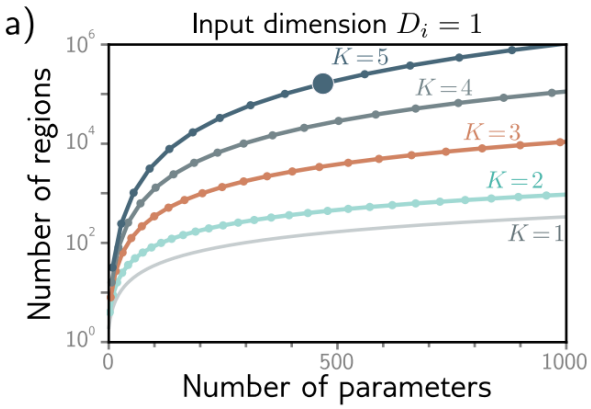


Figure 2.6: Number of linear regions as a function of the number of parameters used and the number of layers in the network. Image taken from Prince [5].

same number of parameters, a *deep neural network* is capable of modeling more complex functions than a *shallow neural network*. Although this represents an important theoretical advantage, it should be emphasized that the linear regions produced by a deep network are not completely independent of one another: their structure is constrained by dependencies and symmetries induced by the architecture. Consequently, simply increasing the number of linear regions does not necessarily guarantee improved performance, unless:

- The true function to be approximated exhibits symmetries compatible with those learned by the network.

- The function to be modeled consists of multiple simple functional subcomponents, that is, it can be expressed as a composition of more elementary functions.

An additional advantage of *deep neural networks*, directly related to what has just been discussed, is their greater generalization capability compared to shallow architectures, particularly when making predictions on unseen data during the training phase. However, a potential drawback should also be noted: deep networks tend to have a higher propensity for *overfitting*, especially in scenarios where the amount of available data is limited (Figure 2.7).

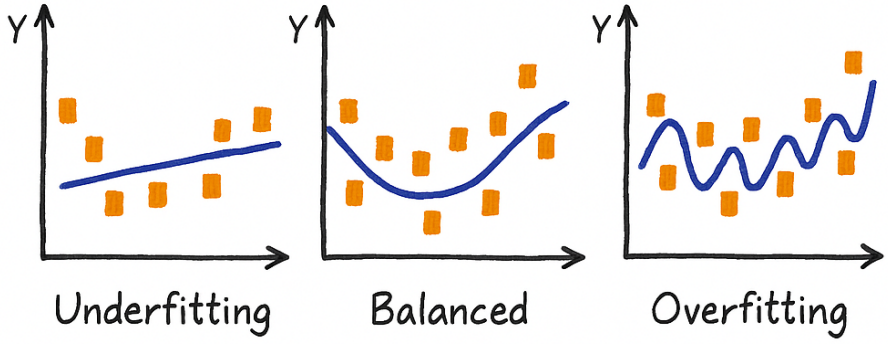


Figure 2.7: Overfitting. Image created by the author.

Loss Function

In the previous subsections, we observed that both *shallow* and *deep* neural networks are described by a set of parameters, denoted by $\vec{w}$ and $\vec{b}$. As these parameters vary, the function represented by the model changes accordingly, and can therefore be viewed as belonging to a family of functions. During the training phase, the objective is to find the parameters that yield the best possible correspondence between input and output with respect to the specific task under consideration. In this paragraph, we clarify what is meant by "best possible mapping." The fundamental idea is that, given a training dataset $\{x_i, y_i\}$, a loss function $L(\vec{w}, \vec{b})$ is defined, which returns a single numerical value measuring the discrepancy between the model predictions $f(x_i, \vec{w}, \vec{b})$ and the corresponding true values y_i. The model that minimizes this loss function is considered to be the one that best approximates the relationship between input and output. Among the various loss functions available in the literature, the most relevant are:

- **Mean Squared Error (MSE):** used for regression tasks. It computes the average of the squared differences between predicted and true values, as follows:

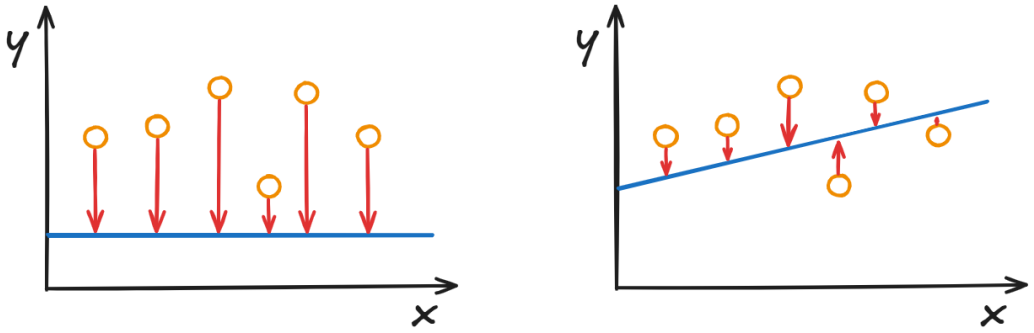
$$MSE = \frac{1}{n} \cdot \sum_{i=1}^{n} (f(x_i, \vec{w}, \vec{b}) - y_i)^2 \tag{2.7}$$

- **Cross-Entropy Loss:** used for classification tasks. It compares the predicted probability distribution $f(x_i, \vec{w}, \vec{b})$ with the true distribution

y_i, encouraging the model to assign higher probabilities to the correct classes:

$$Cross - Entropy = -\sum_{i=1}^{n} y_i \cdot \log(f(x_i, \vec{w}, \vec{b})) \tag{2.8}$$

In the present work, we will focus on regression problems rather than classification tasks, and therefore we will adopt the MSE as the Loss Function.



(a) High MSE value (b) Low MSE value

Figure 2.8: Comparison of the MSE for two linear neural network models. Image created by the author.

Fitting Models

In the previous subsection, we observed how a given neural network model can approximate the input–output pairs of a training dataset $\{x_i, y_i\}$ more or less effectively, depending on the values assumed by its parameters. In this subsection, we focus on the algorithms most commonly used to train a neural network, with the objective of identifying the parameters that minimize the loss function. The first algorithm we examine is Gradient Descent, which consists of two main steps:

1) The first step consists in computing the gradient of the loss function

with respect to the parameters

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_{11}^{[1]}} \\ \frac{\partial L}{\partial w_{12}^{[1]}} \\ \vdots \\ \frac{\partial L}{\partial b_{1}^{[1]}} \\ \vdots \\ \frac{\partial L}{\partial b_{n}^{[K]}} \end{bmatrix} \tag{2.9}$$

2) The second step consists in updating the parameters as follows

$$\begin{bmatrix} \vec{w} \\ \vec{b} \end{bmatrix}^{t+1} = \begin{bmatrix} \vec{w} \\ \vec{b} \end{bmatrix}^{t} - \alpha \cdot \nabla L \tag{2.10}$$

where $\vec{w}$ and $\vec{b}$ are the parameters represented as column vectors and α denotes a value fixed a priori, known as the ***Learning Rate***.

The idea underlying this algorithm can be easily visualized in a one-dimensional case, where we consider a neural network composed of a single input, a single output, and one hidden layer consisting of a single neuron. The adopted activation function $a(\cdot)$ is the identity function. We further assume that the only parameter to be optimized is the weight $\vec{w}$, while the bias b is considered fixed a priori and not subject to learning. In this context, the function implemented by the network reduces to:

$$f(w) = b + w \cdot x \tag{2.11}$$

Consequently, the Mean Squared Error (MSE) takes the form of a parabola with respect to the independent variable w:

$$MSE(w) = \frac{1}{n} \sum_{i=1}^{n} \left(w^2 x_i^2 + 2wbx_i + b^2 + y_i^2 - 2by_i - 2wx_i y_i \right) = a \cdot w^2 + c \cdot w + d \tag{2.12}$$

where the coefficients a, c, and d become known once the dataset $\{x_i, y_i\}$ and the value of b are fixed, and are defined as follows:

$$a = \frac{1}{n}\sum_{i=1}^{n} x_i^2 \tag{2.13}$$

$$c = \frac{1}{n}\sum_{i=1}^{n}(2bx_i - 2x_i y_i) \tag{2.14}$$

$$d = \frac{1}{n}\sum_{i=1}^{n}(b^2 + y_i^2 - 2by_i) \tag{2.15}$$

Figure 2.9 visually illustrates the functioning of the Gradient Descent algorithm applied to the case just described.

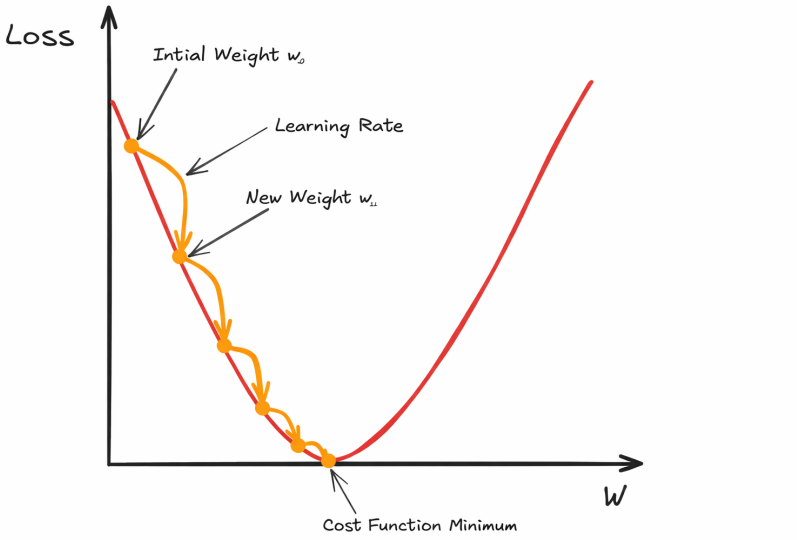
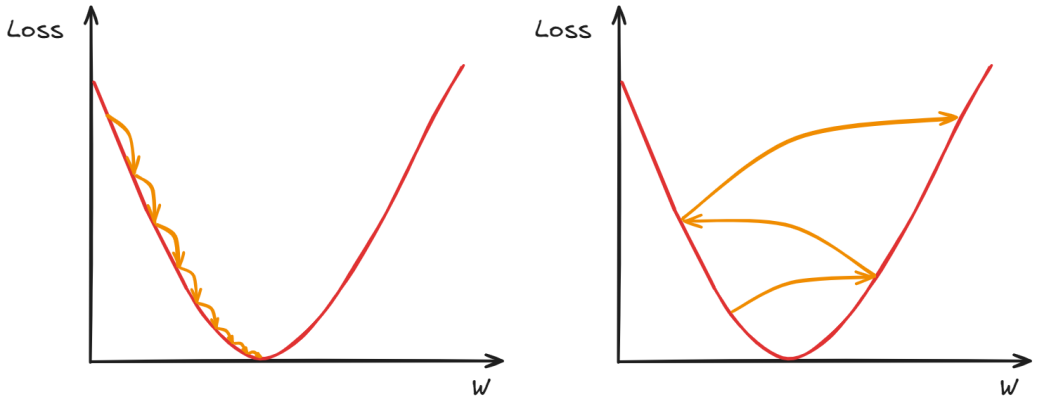


Figure 2.9: Gradient Descent. Image created by the author.

The process begins by randomly assigning an initial value to the weight w and proceeds iteratively, updating the parameter at each step, until the variation of the MSE function between two consecutive iterations (from t to $t+1$) becomes smaller than a predefined threshold. From the figure, it is also possible to observe the crucial role of the parameter α, referred to as the learning rate, which, together with the value of the local derivative, determines the magnitude of the step taken to update the parameter, that is, to move from w^t to w^{t+1}. As shown in Figure 2.10, the value of α affects both the convergence speed of the algorithm and its stability: a value that is too small leads to very slow convergence, whereas an excessively large value may prevent the algorithm from converging. In practice, the values of α are

often chosen empirically, based on preliminary experiments. The algorithm described above proves particularly effective when the loss function is convex, since, once a reasonable learning rate is selected, it guarantees convergence to the global minimum of the function. However, in real neural networks, loss functions are typically non-convex due to the use of nonlinear activation functions.



(a) Small Learning Rate (b) Large Learning Rate

Figure 2.10: Comparison of two Learning Rate values. Image created by the author.

Non-convexity introduces the risk that the algorithm, depending on the parameter initialization, converges to a local minimum rather than to the global minimum. This behavior is illustrated in Figure 2.11: if the parameters are initialized at point A, the algorithm converges to the global minimum; if instead the initialization occurs at point B, convergence takes place toward a local minimum. To address this issue, a common alternative is represented by the **Stochastic Gradient Descent** (SGD) algorithm. The idea behind this method is simple: at each iteration, instead of computing the gradient over the entire dataset, a random subset of examples (called a minibatch or batch) is selected, and the gradient of the loss function is computed only on this subset.
From a certain perspective, this is equivalent to performing gradient descent on a different loss function at each iteration, since the loss depends both on the model and on the selected data. Consequently, the function continuously changes, making the algorithm more "noisy," but potentially capable of escaping local minima. Despite this variability, the expected value of the loss and of the gradient at each point remains the same as in the deterministic classical gradient descent approach. The last algorithm we consider is

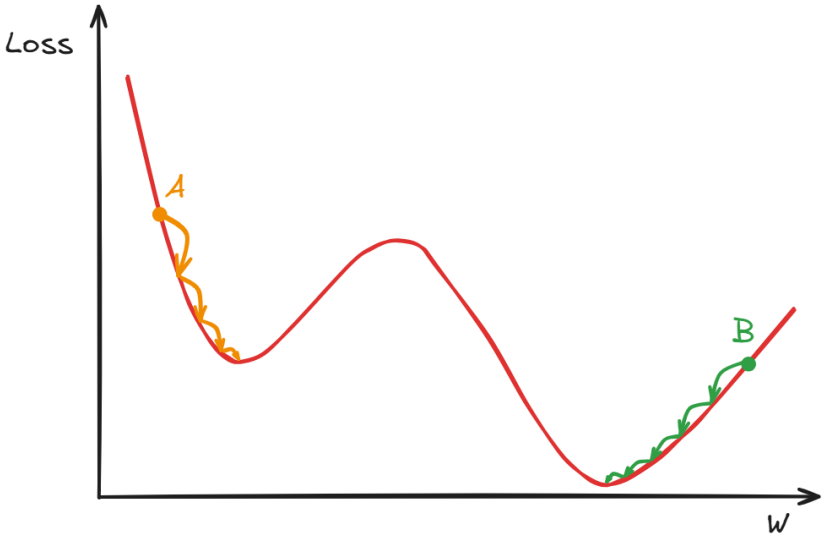


Figure 2.11: Non-convex function. Image created by the author.

Adaptive Moment Estimation (ADAM). This is a more advanced method that introduces a learning rate adaptation mechanism: instead of keeping it constant, ADAM dynamically adjusts the update step for each parameter, taking into account both the mean of the gradients and their variance.

Difference Between Hyperparameters and Parameters

We conclude this section by briefly clarifying the difference between **parameters** and **hyperparameters** within a *machine learning* model, with particular reference to neural networks. **Parameters** are the variables automatically learned by the model during the training phase and depend directly on the training data. In the case of the neural networks discussed previously, these parameters are represented by the weight vectors $\vec{\mathbf{w}}$ and bias vectors $\vec{\mathbf{b}}$. **Hyperparameters**, on the other hand, are variables that are *not learned* during training, but are chosen *before* the learning process begins. Their selection can be performed manually or through automatic hyperparameter optimization techniques. These values influence the behavior and performance of the learning algorithm. We can distinguish two main categories of hyperparameters:

- **Architectural hyperparameters:** they control the structure of the neural network. Typical examples include the number of neurons in the hidden layers (*hidden layers*), the total number of layers, and the activation function employed.

- **Optimization hyperparameters:** they regulate the functioning of the learning algorithm. Examples include the value of the *learning rate*, the *batch* size, and the total number of epochs (*epochs*) or iterations.

2.2 Graphs

Graphs are mathematical structures that are extremely useful for representing and analyzing complex systems. In their most general definition, they consist of a set of objects called **nodes** (or **vertices**) connected to one another through elements known as **edges**. The expressive power of graph-based representations lies in the fact that they are not limited to describing the individual properties of nodes, but are also capable of highlighting the *relationships* between them. A mathematical analysis conducted on such structures can therefore provide a deep understanding of complex real-world systems, which would be difficult (or even impossible) to achieve using more traditional models. Some concrete examples of real systems that can be effectively modeled using graphs include:

- **Social Networks:** nodes represent individuals, while edges represent social relationships (such as friendship) between two people.
- **Molecules:** nodes correspond to atoms, and edges represent the chemical bonds between them.
- **Computational Meshes:** nodes represent the points of the mesh, while edges may indicate geometric distances or, in the context of physical simulations such as computational aerodynamics, physical quantities exchanged between two nodes (for example, momentum).

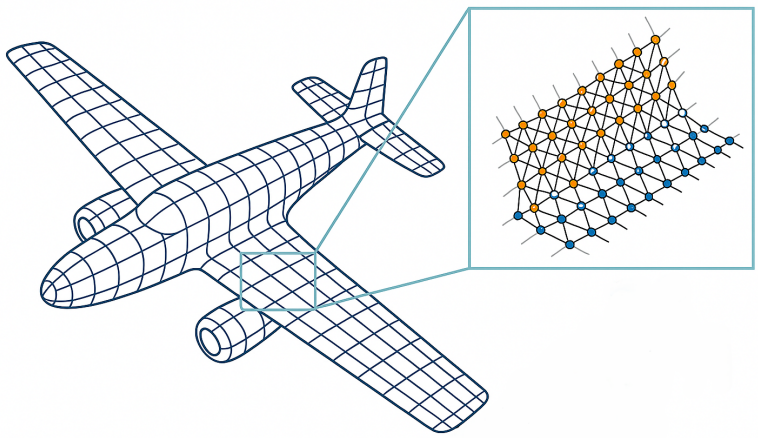


Figure 2.12: Computational Mesh. Image created by the author.

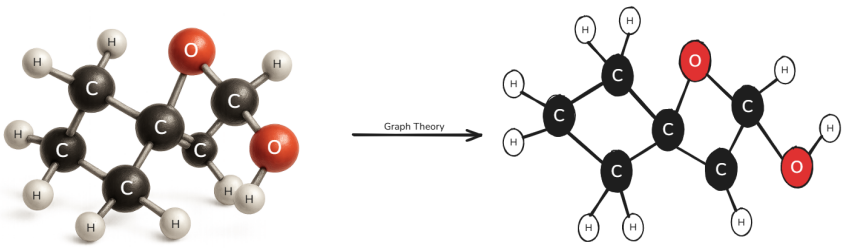


Figure 2.13: Molecule. Image created by the author.

Figure 2.14: Social Network. Image created by the author.

Introduction to Graph Theory

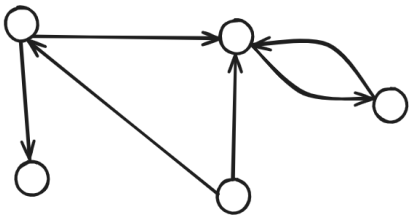
Formally, a graph is defined as a set of nodes or vertices connected to each other by edges

$$\mathbf{G} = (\mathbf{V}, \mathbf{E}) \tag{2.16}$$

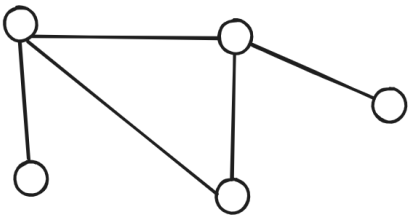
An edge connecting a node $u \in \mathbf{V}$ to a node $v \in \mathbf{V}$ is defined as $(u, v) \in \mathbf{E}$. A graph can be categorized according to its structural properties and the nature of the connections between vertices. Among the different types of graphs, we find:

- **Directed and undirected graphs:** a graph is said to be undirected when $(u, v) = (v, u)$, that is, when an edge connecting a node u to a

node v is identical to an edge connecting node v to node u. Conversely, it is defined as directed when $(u, v) \neq (v, u)$.



(a) Directed graph.



(b) Undirected graph

- **Weighted and unweighted graphs:** a graph is defined as weighted when edges have an associated weight that may vary from one edge to another, representing for example a distance, the strength of a given connection, and so on. It is defined as unweighted when edges do not carry a weight, meaning that they simply represent the presence or absence of a connection.

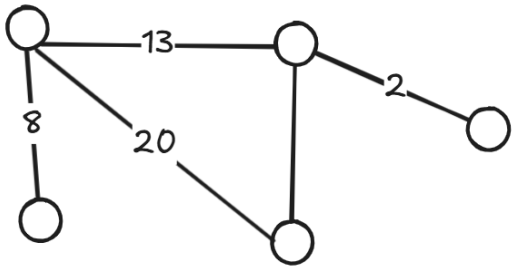


Figure 2.16: Weighted graph. Image created by the author.

- **Simple graphs and multigraphs:** a graph is defined as simple when there are no edges connecting a vertex to itself and when there is at most one edge connecting the same pair of vertices. A multigraph, on the other hand, allows multiple edges between the same pair of vertices, enabling the representation of different types of interactions between two vertices.

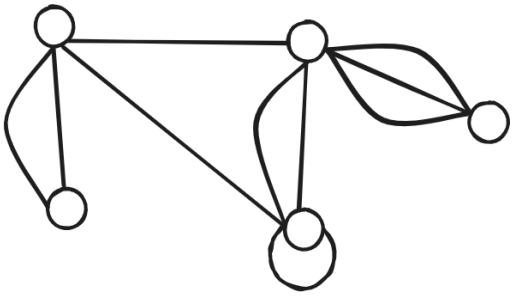


Figure 2.17: Multigraph. Image created by the author.

- **Complete graph:** a graph is defined as complete when every pair of distinct vertices is connected by exactly one edge.

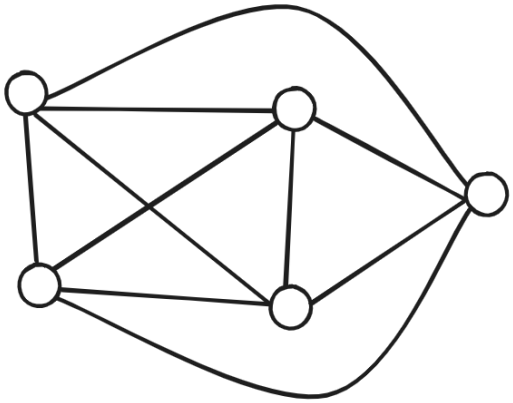


Figure 2.18: Complete graph. Image created by the author.

- **Bipartite graph:** a graph is defined as bipartite when the set of vertices $\mathbf{V}$ can be divided into two subsets $\mathbf{V}_1$ and $\mathbf{V}_2$, such that every edge connects a vertex from one subset to a vertex belonging to the other subset.

Mathematical Representation of Graphs

A graph can be represented mathematically in multiple ways, depending on the type of analysis or application. In this subsection, we analyze some of the most common ways of representing both the structure of a graph and its properties, which can provide deeper insight into its organization. Among the most common mathematical objects used to describe the structure of a graph, we find:

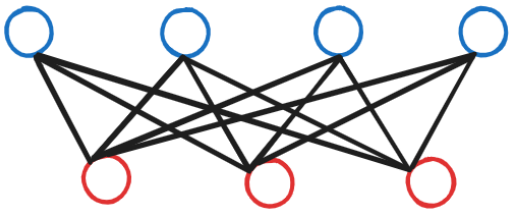


Figure 2.19: Bipartite graph. Image created by the author.

- **Adjacency Matrices:** this represents one of the most common ways to describe a graph. Given a graph **G** composed of a total number of n nodes, the adjacency matrix $[\mathbf{A}]$ is an $n \times n$ matrix representing the presence or absence of an edge between a pair of vertices. The matrix element $[\mathbf{A}]_{ij}$, in the case of unweighted graphs, is defined as:

$$[\mathbf{A}]_{ij} = \begin{cases} 1 & \text{if there is an edge between vertices } v_i \text{ and } v_j, \\ 0 & \text{otherwise} \end{cases} \tag{2.17}$$

  In the case of weighted graphs, the matrix element $[\mathbf{A}]_{ij}$ is defined as:

$$[\mathbf{A}]_{ij} = \begin{cases} w_{ij} & \text{if there is an edge between vertices } v_i \text{ and } v_j, \\ 0 & \text{otherwise} \end{cases} \tag{2.18}$$

  where w_{ij} represents the weight value of the edge connecting vertices v_i and v_j. From the given definition of the adjacency matrix, it is straightforward to observe that in the case of an undirected graph, the matrix $[\mathbf{A}]$ is symmetric. Conversely, if the graph is directed, the matrix $[\mathbf{A}]$ is not necessarily symmetric. Finally, in the case of a multigraph, the mathematical object $[\mathbf{A}]$ is no longer a matrix but a tensor, known as the adjacency tensor $[\mathbf{A}]$, of dimension $n \times n \times m$, where n represents the number of nodes and m represents the total number of properties associated with each edge.

- **Incidence Matrices:** these are used to capture the relationship between vertices and edges. The incidence matrix $[\mathbf{B}]$ is an $n \times m$ matrix, where n represents the number of vertices and m represents the number of edges, such that each row corresponds to a vertex and each column corresponds to an edge. The matrix element $[\mathbf{B}]_{ij}$ is defined as:

$$[\mathbf{B}]_{ij} = \begin{cases} 1 & \text{if vertex } v_i \text{ is incident to edge } e_j, \\ 0 & \text{otherwise} \end{cases} \tag{2.19}$$

Among the most important graph properties, we find:

- **Degree of a vertex:** this represents the number of edges connected to a given vertex. In an undirected graph, the degree of a vertex v_i, denoted as $d(v_i)$, is simply the number of edges incident to v_i. Mathematically, this can be expressed as:

$$d(v_i) = \sum_{j=1}^{n} [\mathbf{A}]_{ij} \tag{2.20}$$

  In the case of a directed graph, one can define an in-degree $d_{in}(v_i)$ and an out-degree $d_{out}(v_i)$, representing respectively the incoming and outgoing edges of node v_i. Mathematically, this can be expressed as:

$$d_{in}(v_i) = \sum_{j=1}^{n} [\mathbf{A}]_{ji} \tag{2.21}$$

$$d_{out}(v_i) = \sum_{j=1}^{n} [\mathbf{A}]_{ij} \tag{2.22}$$

  It is also worth noting that there exists a matrix whose diagonal elements correspond to the degrees of the respective vertices, known as the degree matrix.

- **Diameter and radius of a graph:** to explain the diameter of a graph, it is first necessary to introduce the concept of a path. A path in a graph is a sequence of edges connecting a sequence of distinct vertices. If the first and last vertices of the path coincide, the path is called a cycle. The diameter of a graph $\mathbf{G}$ can be defined as:

$$\text{diam}(\mathbf{G}) = \max_{u,v \in V} d(u, v) \tag{2.23}$$

  where $d(u, v)$ represents the shortest path connecting two vertices $u, v \in V$. The radius can be defined as:

$$\text{rad}(\mathbf{G}) = \min_{u \in V} e(u) \tag{2.24}$$

  where $e(u)$ represents the eccentricity of a vertex u, defined as the greatest distance from u to any other vertex.

- **Vertex centrality:** centrality is an important property of a vertex because it allows us to measure how important a vertex is within the

graph. Naturally, the importance of a node depends on what is meant by "important." For this reason, there are different ways to define centrality. A simple way to measure centrality is through **degree centrality**, which increases proportionally to the number of edges connected to a vertex. Other centrality measures, which will not be analyzed in detail in this work but are worth mentioning, include **Closeness Centrality**, **Betweenness Centrality**, and **Eigenvector Centrality**.

2.3 Graph Neural Network

In this paragraph, we explain how neural networks can be integrated with structures such as graphs in order to obtain three main types of predictions:

- **Graph-level predictions:** the objective of this type of prediction is to infer one or more properties of an entire graph. A representative example is the following: given the structure of a molecule, transformed into a graph, we aim to determine the smell associated with that molecule.

- **Edge-level predictions:** the objective of this type of prediction is either to determine whether a specific edge exists between two nodes or to predict the properties associated with a given edge.

- **Node-level predictions:** the objective of this type of prediction is to infer the properties of a specific node.

In the present work, the GNN model employed focuses on node-level predictions. The key mechanism used by these networks to perform such predictions is known as **message passing** or **neighborhood aggregation**. This mechanism allows nodes to iteratively exchange information with their neighboring nodes, updating their properties at each iteration.

Message Passing

The general formula describing the message passing process can be summarized in a single expression as follows:

$$\mathbf{x}_i^{(k+1)} = \gamma^{(k)} \left(\mathbf{x}_i^{(k)}, \bigoplus_{j \in \mathcal{N}(i)} \phi^{(k)} \left(\mathbf{x}_i^{(k)}, \mathbf{x}_j^{(k)}, \mathbf{e}_{ij} \right) \right) \tag{2.25}$$

We now examine each individual component appearing in the equation that describes the message passing mechanism:

- $\mathbf{x}_i^{(k)}$: represents the feature vector of node (i) evaluated at iteration (k) of the GNN. Before the iterative process begins, these features may represent intrinsic properties of the node; for example, in a computational fluid dynamics application, they may include velocity, pressure, density, and temperature. As the iterations proceed, the feature vector incorporates information from neighboring nodes and from the graph structure itself.

- $\mathcal{N}(\mathbf{i})$: represents the set of neighboring nodes of node (i). During the first iterations, the node interacts only with its immediate neighbors; however, as the number of iterations increases, the node becomes indirectly influenced by more distant nodes in the graph, eventually capturing global structural information.

- $\boldsymbol{\phi}^{(k)}$: represents the **message function**. This function outputs the message that each neighboring node j sends to node i. Its inputs are the feature vector of the source node j, the feature vector of the target node i, and the edge feature vector $\mathbf{e}_{ij}$ connecting nodes i and j. In our case, this function is implemented as a neural network of the MLP type.

- $\bigoplus$: represents the aggregation operation used to combine all messages m_{ij} received from neighboring nodes $j \in \mathcal{N}(i)$. Common choices for this operation include:

  1. **Summation**:
  $$\sum_{j \in \mathcal{N}(i)} m_{i,j} \tag{2.26}$$

  2. **Mean**:
  $$\frac{1}{|\mathcal{N}(i)|} \cdot \sum_{j \in \mathcal{N}(i)} m_{i,j} \tag{2.27}$$

  3. **Max pooling**:
  $$\max_{j \in \mathcal{N}(i)} m_{i,j} \tag{2.28}$$

- $\gamma^{(k)}$: represents the **update function**. This function takes as input both the aggregated messages from neighboring nodes and the current node feature vector $\mathbf{x}_i^{(k)}$, and outputs the updated node feature vector $\mathbf{x}_i^{(k+1)}$ for the next layer.

This procedure is performed for several iterations. In particular, at each iteration, every node aggregates information from its adjacent nodes and, as the iterations progress, its feature vector incorporates information from nodes that are increasingly distant in the graph. In general, after K iterations, the feature vector of each node contains information from nodes located up to K hops away. At this point, one may ask what type of information each node is actually able to acquire. In general, this information can be divided into two main categories:

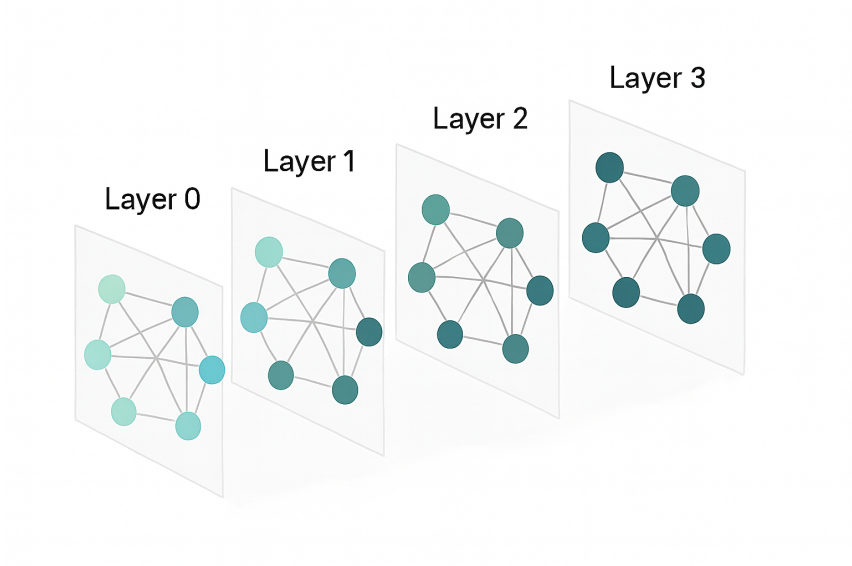


Figure 2.20: Message passing. Adapted from [7].

- **Structural information**: for example, after K iterations, the feature vector of a node u may encode information related to the **degree** of nodes within K hops in the graph.
- **Node feature information**: for example, after K iterations, the feature vector of a node u may include information about the **features** of nodes in its K-hop neighborhood. This type of local aggregation can be seen as analogous to what occurs in Convolutional Neural Networks (CNNs), where convolutional filters aggregate information from small spatial regions of an image.

Although this structure is highly effective in capturing complex dependencies between graph nodes, an excessive number of iterations K may lead to the problem of **over-smoothing**, in which node feature vectors become indistinguishable from one another, thereby reducing the model's ability to discriminate between different nodes.

Advantages of Using Graph Neural Networks

We conclude this paragraph by analyzing some of the main advantages offered by architectures based on **Graph Neural Networks** (GNNs), compared to other well-established architectures in the field of machine learning, such as **Convolutional Neural Networks** (CNNs). Among the main advantages, we can identify:

- **Processing of unstructured data**: Although Convolutional Neural Networks (CNNs) are particularly effective in analyzing structured data, such as images, where each pixel has a fixed and regular number of neighbors, Graph Neural Networks (GNNs) offer the advantage of operating on unstructured data. In particular, they are capable of handling input structures in which relationships between nodes do not follow a fixed pattern, successfully aggregating information from nodes connected in an irregular and heterogeneous manner.

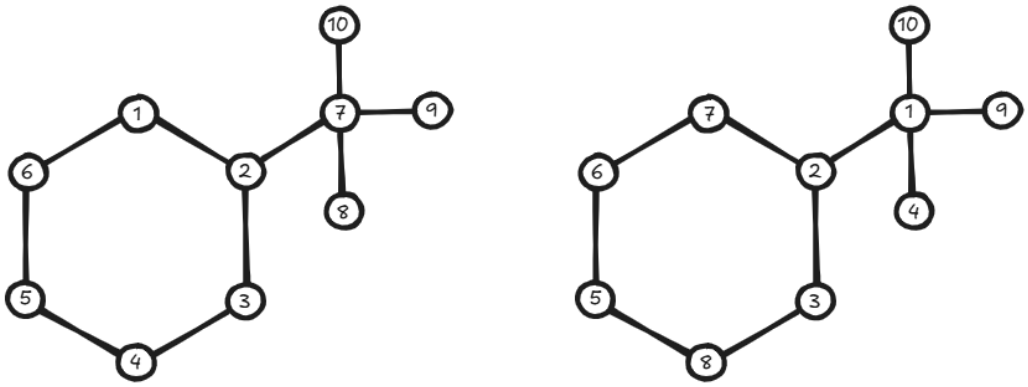
- **Permutation invariance and equivariance**: before explaining why this is a fundamental property of Graph Neural Networks (GNNs), it is useful to clarify what is meant by **permutation invariance** and **permutation equivariance** in the context of graphs. In mathematical terms, a function f that takes an adjacency matrix $\mathbf{A}$ as input is said to be **permutation invariant** if it satisfies the following relation:

$$f(\mathbf{P}\mathbf{A}\mathbf{P}^\top) = f(\mathbf{A}) \tag{2.29}$$

It is said to be **permutation equivariant** if:

$$f(\mathbf{P}\mathbf{A}\mathbf{P}^\top) = \mathbf{P}f(\mathbf{A}) \tag{2.30}$$

where $\mathbf{P}$ is a permutation matrix. In the context of GNNs, permutation invariance is fundamental when dealing with **graph-level tasks**, where the objective is to assign a label or a property to the entire graph (as in the case of molecular classification). Equivariance, on the other hand, is crucial in **node-level tasks**, where the model produces an output for each node. In this latter case, if the input nodes are permuted, the outputs must be permuted accordingly. To better understand the meaning of permutation invariance, consider Figures 2.21.a and 2.21.b. They represent two graphs describing physically the same molecule, but differing in the ordering of the nodes.



(a) Graph A (b) Graph B

Figure 2.21: Comparison between two graphs representing the same molecule. Image created by the author.

The adjacency matrix of Graph A is:

$$\mathbf{A_a} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{2.31}$$

The adjacency matrix of Graph B is:

$$\mathbf{A_b} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.32}$$

Although $\mathbf{A_a}$ and $\mathbf{A_b}$ are different, they represent the same graph up to a permutation of the nodes. In a graph classification task, for example predicting the *smell* of a molecule, it is necessary that the network output be **invariant** with respect to node permutations. Unlike GNNs, a structure such as a Convolutional Neural Network (CNN) would not be suitable in this context, since it is designed to be **invariant to spatial translations**, a property effective for structured data such as images, where each pixel has well-defined neighbors. However, CNNs are not capable of handling **arbitrary permutations** of input elements, as occurs in graphs, where node ordering has no absolute meaning. For this reason, GNNs provide a more general and effective solution for problems involving irregular data structures.

- **Graphs of variable size**: A significant limitation of architectures such as Convolutional Neural Networks (CNNs) is their inability to process inputs of variable size. Due to their structure, these networks require fixed-size input data: for example, a CNN designed for 128×128 pixel images cannot be directly applied to 256×256 pixel images without modifying the network architecture. Graph Neural Networks (GNNs), on the other hand, are capable of handling graphs of varying sizes, from a few nodes to thousands, since they operate in a **local** manner, aggregating information in the neighborhood of each node rather than processing the entire graph globally. However, although this characteristic provides significant flexibility, an important open question remains: it is not yet fully understood how a GNN trained on small graphs (for example, with 10 nodes) behaves when applied to much larger graphs containing hundreds or thousands of nodes. In such cases, performance degradation phenomena may emerge, depending on the network architecture.

2.4 Neural Operator

In the previous section, we described how neural networks can classically be used to learn mappings between finite-dimensional Euclidean spaces. In this section, we introduce a generalization of neural networks, namely **Neural Operators**, which are capable of learning operators that map between infinite-dimensional functional spaces. These recently introduced models exhibit two fundamental properties, analyzed in the reference works [2] and [3]. The first is **discretization invariance**, meaning that neural operators are able to:

- Act on any discretization of the input function, that is, they accept any set of points in the input domain
- Be evaluated at any point in the output domain
- Converge to a continuous operator as the discretization is progressively refined.

These conditions clarify why **GNNs**, although satisfying the first two, cannot be considered discretization invariant because, as discussed in the previous paragraph, progressively refining the grid typically does not lead to test error convergence to zero, thus violating the third condition. The second property is that of being **universal approximators**: under suitable architectural and regularity assumptions, neural operators are capable of approximating continuous operators acting between Banach spaces with arbitrary precision. Neural operators present significant application potential for two main reasons:

- In the context of physical modeling, many phenomena are described by partial differential equations (PDEs). The purpose of neural operators in this setting is to learn a surrogate of the inverse operator, thereby reducing computational costs compared to traditional numerical methods. To better clarify this concept, consider for example a family of PDEs of the form

$$\begin{cases} (L_a u)(x) = f(x), & x \in D, \\ u(x) = 0, & x \in \partial D. \end{cases} \tag{2.33}$$

  where L_a represents an operator depending on the function a. A natural operator derived from this PDE is

$$G^\dagger := L_a^{-1} f \colon A \to U, \qquad a \mapsto u. \tag{2.34}$$

$G^\dagger$ therefore represents an operator that associates to a function a the corresponding solution function u. The objective of a Neural Operator in this case is to approximate the operator $G^\dagger$ by constructing a parametric map

$$G_\theta : A \to U, \qquad \theta \in \mathbb{R}^p, \tag{2.35}$$

whose parameters belong to the finite-dimensional space $\mathbb{R}^p$, and selecting $\theta^\dagger \in \mathbb{R}^p$ such that $G_{\theta^\dagger} \approx G^\dagger$.

- In data-driven scenarios where an accurate physical model is incomplete or unavailable, Neural Operators allow the direct learning of operator-to-operator mappings between functional spaces from data.

Architecture of a Neural Operator

The complete structure of a neural operator can be divided into three main blocks:

- **Lifting**: using a pointwise function $\mathbb{R}^{d_a} \to \mathbb{R}^{d_{v_0}}$, the input $\{a : D \to \mathbb{R}^{d_a}\} \mapsto \{v_0 : D \to \mathbb{R}^{d_{v_0}}\}$ is mapped to its first hidden representation. This operation is referred to as lifting because one typically chooses $d_{v_0} > d_a$. This step is implemented through a fully local operator.

- **Iterative kernel integration**: for $t = 0, \dots, T-1$, each hidden representation is mapped to the next one $\{v_t : D_t \to \mathbb{R}^{d_{v_t}}\} \mapsto \{v_{t+1} : D_{t+1} \to \mathbb{R}^{d_{v_{t+1}}}\}$ through the action of the sum of a local linear operator, a nonlocal integral operator with kernel, and a bias function, followed by a pointwise nonlinearity that is identical across layers.

- **Projection**: using a pointwise function $\mathbb{R}^{d_{v_T}} \to \mathbb{R}^{d_u}$, the final hidden representation $\{v_T : D' \to \mathbb{R}^{d_{v_T}}\} \mapsto \{u : D' \to \mathbb{R}^{d_u}\}$ is mapped to the output function. Similarly to the first block, one typically chooses $d_{v_T} > d_u$; therefore, this is a projection phase also implemented through a fully local operator.

The general structure can be represented in compact form as follows:

$$G_\theta := Q \circ \mathcal{L}_T \circ \cdots \circ \mathcal{L}_1 \circ P \tag{2.36}$$

where Q and P represent the lifting and projection blocks, respectively, while $\mathcal{L}_t$ denotes the kernel integration block indexed by the corresponding layer. The internal structure of the kernel integration block $\mathcal{L}_t$ is defined as follows:

$$v_{t+1}(x) = \sigma\left(\underbrace{W v_t(x)}_{\text{Local operator}} + \underbrace{\int_D \kappa(x,y)\, v_t(y)\, \mathrm{d}y}_{\text{Nonlocal operator}} + \underbrace{b(x)}_{\text{Bias}}\right) \tag{2.37}$$

We conclude this subsection by emphasizing that it is precisely the combination of a nonlocal operator with a nonlinearity that enables neural operators to form architectures capable of learning a broad class of continuous operators [3].

Graph Neural Operator

We conclude this paragraph by introducing a particular class of neural operators, known as the **graph neural operator**. Starting from equation (2.37), which for simplicity does not include the bias term, we observe that, since a computer necessarily operates in finite-dimensional spaces, the first step toward implementing such a formulation consists in approximating the integral through a numerical technique and discretizing the domain with a finite set of points $\{x_1, \ldots, x_J\} \subset D$.

$$v_{t+1}(x) = \sigma\left(W\, v_t(x) \;+\; \frac{1}{J}\sum_{l=1}^{J} \kappa(x, x_l)\, v_t(x_l)\right), \qquad j = 1, \ldots, J. \tag{2.38}$$

This expression highlights the analogy with equation (1.26), which describes the *message passing* mechanism typical of **graph neural networks**, thereby justifying the name **graph neural operator**. In order to reduce the computational cost associated with this model, two fundamental approximations are introduced:

- **Nyström approximation**: this approximation consists in evaluating the summation appearing in equation (2.38) over a reduced number of points J', uniformly distributed and randomly sampled, with $J' \ll J$.
- **Domain truncation**: this approximation consists in restricting the integral to a subdomain of D that depends on the evaluation point $x \in D$. For simplicity and ease of implementation, we consider the subdomain:

  $$s(x) = B(x, r) \cap D, \qquad B(x, r) = \{y \in \mathbb{R}^d : \|y - x\|_{\mathbb{R}^d} < r\} \tag{2.39}$$

  for a fixed radius $r > 0$.

This second approximation has the advantage of preserving invariance with respect to the chosen discretization. Indeed, since the radius r is defined in the physical space, progressively refining the discretization increases the number of nodes included in the subdomain, thereby maintaining the consistency of the model.

Bibliography

[1] William L Hamilton. *Graph representation learning*. Morgan & Claypool Publishers, 2020.

[2] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

[3] Samuel Lanthaler, Zongyi Li, and Andrew M Stuart. Nonlocality and nonlinearity implies universality in operator learning. *Constructive Approximation*, pages 1–43, 2025.

[4] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.

[5] Simon JD Prince. *Understanding deep learning*. MIT press, 2023.

[6] Michele Quattromini. *Graph Neural Networks for fluid mechanics: data-assimilation and optimization*. PhD thesis, Université Paris-Saclay; Politecnico di Bari, 2024.

[7] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B Wiltschko. A gentle introduction to graph neural networks. *Distill*, 6(9):e33, 2021.

[8] Gilad Yehudai, Ethan Fetaya, Eli Meirom, Gal Chechik, and Haggai Maron. From local structures to size generalization in graph neural networks. In *International Conference on Machine Learning*, pages 11975–11986. PMLR, 2021.

Chapter 3

The Governing Equations of Fluid Dynamics

This chapter presents the governing equations of fluid dynamics that constitute the theoretical foundation of all the numerical simulations performed in this work. A rigorous understanding of these equations is essential to ensure that the numerical results remain consistent with the underlying physical principles. The inclusion of this chapter is motivated by two main reasons:

1. All the CFD simulations developed in this thesis are based on these governing equations. Therefore, evaluating the physical consistency and reliability of the numerical results requires a clear understanding of their mathematical and physical structure.

2. The governing equations can be expressed in different mathematical forms (integral or differential, conservative or non-conservative). While these formulations are theoretically equivalent under appropriate assumptions, the choice of a specific form plays a crucial role in numerical implementations. In CFD algorithms, different formulations may lead to variations in conservation properties, numerical stability, and accuracy.

The governing equations represent the mathematical formulation of three fundamental physical principles upon which fluid dynamics is based:

- **Conservation of Mass**
- **Conservation of Momentum**
- **Conservation of Energy**

To derive these equations, it is necessary to introduce a model for describing the flow field. In this work, the finite control volume approach is adopted. A control volume is a finite region of space through which fluid flows, bounded by a closed control surface. The control volume may be either fixed or moving; however, in the present derivation, we consider a control volume fixed in space, with fluid flowing across its boundaries. The finite control volume formulation is particularly suited for numerical discretization, as it enforces the conservation laws in integral form over discrete subdomains of the computational domain. By analyzing the balance of mass, momentum, and energy within this finite region, it becomes possible to describe the evolution of the flow in terms of fluxes crossing the control surface and sources acting within the volume. Let us therefore consider a finite control volume drawn within the flow field, fixed in space, with fluid entering and leaving through its boundary, as illustrated in Figure 3.1.

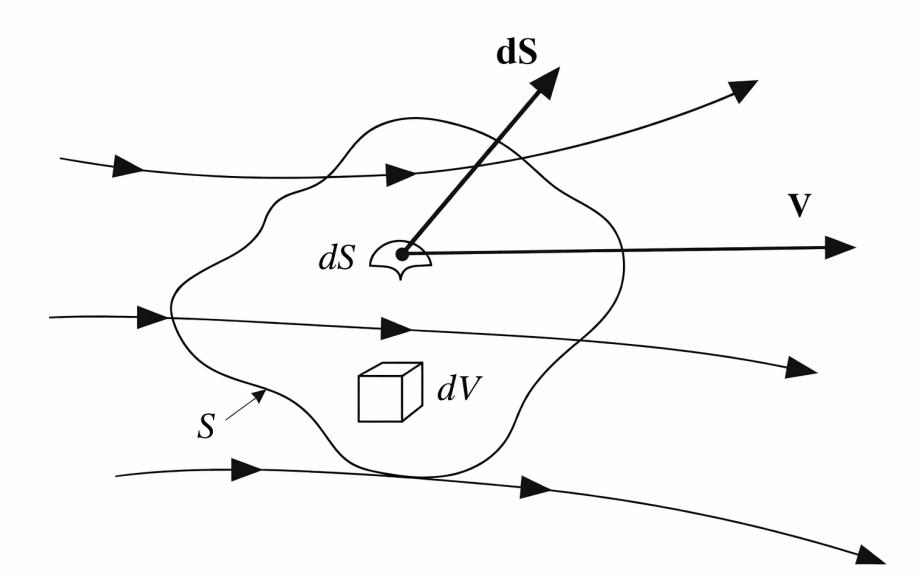


Figure 3.1: Fixed finite control volume. Image taken from Anderson [2]

3.1 Conservation of Mass

The conservation of mass principle, in the absence of sources or sinks, states that **the rate of variation of the mass inside a control volume must be equal to the negative of the net mass flux through its boundary**. The total mass contained within the control volume V is defined as:

$$\int_V \rho \, dV \tag{3.1}$$

and the total mass flux across the control surface S as:

$$\int_S \rho \, \mathbf{V} \cdot \mathbf{n} \, dS, \tag{3.2}$$

the **integral form** of the conservation of mass can be written as:

$$\frac{d}{dt} \int_V \rho \, dV = - \int_S \rho \, \mathbf{V} \cdot \mathbf{n} \, dS. \tag{3.3}$$

if the control volume is fixed in space and time, the volume V does not depend on time, and the time derivative can be moved inside the integral:

$$\int_V \frac{\partial \rho}{\partial t} \, dV = - \int_S \rho \, \mathbf{V} \cdot \mathbf{n} \, dS. \tag{3.4}$$

applying the divergence theorem to the flux term, we obtain:

$$\int_S \rho \, \mathbf{V} \cdot \mathbf{n} \, dS = \int_V \nabla \cdot (\rho \mathbf{V}) \, dV. \tag{3.5}$$

substituting into the previous equation yields:

$$\int_V \left(\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{V}) \right) dV = 0. \tag{3.6}$$

since the control volume is arbitrary, the integrand must be zero at every point in the domain. Therefore, the **differential form** of the conservation of mass is obtained as:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{V}) = 0. \tag{3.7}$$

3.2 Conservation of Momentum

The conservation of momentum principle states that **the rate of variation of the momentum inside a control volume must be equal to the negative of the net momentum flux through its boundary plus the total external forces acting on the control volume**. The total momentum contained within the control volume V is defined as:

$$\int_V \rho \mathbf{V}\, dV \tag{3.8}$$

and the total momentum flux across the control surface S as:

$$\int_S \rho \mathbf{V}(\mathbf{V}\cdot\mathbf{n})\, dS, \tag{3.9}$$

we can write:

$$\frac{d}{dt}\int_V \rho \mathbf{V}\, dV = -\int_S \rho \mathbf{V}(\mathbf{V}\cdot\mathbf{n})\, dS + \mathbf{F}_{\text{ext}}. \tag{3.10}$$

The external forces acting on the control volume can be classified into two types:

- **Body forces**: forces acting throughout the volume of the fluid, such as gravity.
- **Surface forces**: forces acting on the boundary of the control volume, such as pressure and viscous stresses.

The body force contribution can be written as:

$$\mathbf{F}_b = \int_V \rho \mathbf{f}\, dV, \tag{3.11}$$

where $\mathbf{f}$ represents the force per unit mass. The surface force can be written as:

$$\mathbf{F}_s = \int_S -p\,[\mathbf{I}]\cdot\mathbf{n}\, dS + \int_S [\boldsymbol{\tau}]\cdot\mathbf{n}\, dS. \tag{3.12}$$

where p represents the pressure and $[\boldsymbol{\tau}]$ represents the viscous stress tensor. The integral form of the conservation of momentum can therefore be written as:

$$\frac{d}{dt}\int_V \rho \mathbf{V}\, dV + \int_S \rho \mathbf{V}(\mathbf{V}\cdot\mathbf{n})\, dS = \int_V \rho \mathbf{f}\, dV + \int_S (-p\,[\mathbf{I}] + [\boldsymbol{\tau}])\cdot\mathbf{n}\, dS \tag{3.13}$$

if the control volume is fixed in space and time, the volume V does not depend on time, and the time derivative can be moved inside the integral:

$$\int_V \frac{\partial}{\partial t}(\rho \mathbf{V})\, dV + \int_S \rho \mathbf{V}(\mathbf{V}\cdot\mathbf{n})\, dS = \int_V \rho \mathbf{f}\, dV + \int_S (-p[\mathbf{I}] + [\boldsymbol{\tau}])\cdot \mathbf{n}\, dS \tag{3.14}$$

applying the divergence theorem to the surface integrals, we obtain:

$$\int_V \left[\frac{\partial}{\partial t}(\rho \mathbf{V}) + \nabla\cdot\left(\rho\, \mathbf{V}\mathbf{V}\right) - \rho\mathbf{f} - \nabla\cdot(-p[\mathbf{I}] + [\boldsymbol{\tau}])\right] dV = 0. \tag{3.15}$$

since the control volume is arbitrary, the integrand must vanish at every point in the domain. Therefore, the **differential form** of the conservation of momentum is obtained as:

$$\frac{\partial}{\partial t}\left(\rho \mathbf{V}\right) + \nabla\cdot\left(\rho\, \mathbf{V}\mathbf{V}\right) = -\nabla p + \nabla\cdot[\boldsymbol{\tau}] + \rho\mathbf{f}. \tag{3.16}$$

3.3 Conservation of Energy

The conservation of energy principle states that **the rate of variation of the total energy inside a control volume is equal to the negative of the net energy flux through its boundary plus the rate of work done by external forces and the net heat flux through the control surface**. In the present formulation, no volumetric heat sources or internal mechanical work inputs are considered. The total energy per unit mass can be written as:

$$E = e + \frac{V^2}{2} \tag{3.17}$$

where:

- e is the **internal energy** per unit mass, accounting for the microscopic random motion of atoms and molecules within the fluid;
- $\frac{V^2}{2}$ is the **kinetic energy** per unit mass associated with the macroscopic motion of the fluid.

The total energy contained within the control volume V is defined as:

$$\int_V \rho E \, dV = \int_V \rho \left(e + \frac{V^2}{2} \right) dV \tag{3.18}$$

and the total energy flux across the control surface S as:

$$\int_S \rho E (\mathbf{V} \cdot \mathbf{n}) \, dS. \tag{3.19}$$

the rate of work done by external forces can be written as:

$$\dot{W}_b = \int_V \rho \mathbf{f} \cdot \mathbf{V} \, dV \tag{3.20}$$

$$\dot{W}_s = - \int_S p (\mathbf{V} \cdot \mathbf{n}) \, dS + \int_S ([\boldsymbol{\tau}] \cdot \mathbf{n}) \cdot \mathbf{V} \, dS \tag{3.21}$$

the net heat flux through the control surface is denoted by:

$$Q = \int_S \mathbf{q} \cdot \mathbf{n} \, dS \tag{3.22}$$

the integral form of the conservation of energy can therefore be written as:

$$\frac{d}{dt} \int_V \rho E \, dV + \int_S \rho E (\mathbf{V} \cdot \mathbf{n}) \, dS = \dot{W}_b + \dot{W}_s + Q. \tag{3.23}$$

3.4 Closure of the governing equation

The system of conservation equations derived in the previous sections consists of five independent equations: one scalar equation from the conservation of mass, three scalar equations obtained from the vectorial conservation of momentum, and one scalar equation from the conservation of energy. However, these equations involve a larger number of unknown variables. In particular, the set of unknowns includes:

- the density ρ,
- the three velocity components v_x, v_y, v_z,
- the internal energy e,
- the pressure p,
- the six independent components of the viscous stress tensor $[\boldsymbol{\tau}]$,
- the three components of the heat flux vector $\mathbf{q}$.

In total, this leads to fifteen unknown variables. Since the number of unknowns exceeds the number of governing equations, the system is not closed. In order to obtain a mathematically well-posed and solvable system, additional relations must be introduced to link these variables. These additional relations are known as *constitutive equations*. They describe the physical behavior of the fluid and provide the necessary closure to the governing equations. In particular:

- **Newtonian fluid hypothesis (viscous stresses)**: For a Newtonian fluid, and under the Stokes hypothesis, the viscous stress tensor is assumed to be linearly proportional to the rate of deformation tensor. Introducing the velocity gradient $\nabla\mathbf{V}$, the constitutive relation can be written as:

$$[\boldsymbol{\tau}] = \mu\left(\nabla\mathbf{V} + (\nabla\mathbf{V})^T\right) - \frac{2}{3}\mu\left(\nabla\cdot\mathbf{V}\right)[\mathbf{I}], \tag{3.24}$$

  where μ is the dynamic viscosity. For incompressible flows, $\nabla\cdot\mathbf{V} = 0$, and the relation simplifies to:

$$[\boldsymbol{\tau}] = \mu\left(\nabla\mathbf{V} + (\nabla\mathbf{V})^T\right). \tag{3.25}$$

- **Fourier's law (heat flux)**: The heat flux vector is related to the temperature gradient through Fourier's law of heat conduction:

$$\mathbf{q} = -k\nabla T, \tag{3.26}$$

where k is the thermal conductivity and T is the temperature field.

- **Equation of state (thermodynamic closure)**: A further relation is required to link thermodynamic variables such as pressure, density, and internal energy. In general, this can be expressed as an equation of state:

$$p = p(\rho, e), \tag{3.27}$$

together with a caloric relation, for example:

$$T = T(\rho, e). \tag{3.28}$$

For a calorically perfect ideal gas, these relations take the common form:

$$p = \rho RT, \tag{3.29}$$

$$e = c_v T, \tag{3.30}$$

where R is the specific gas constant and c_v is the specific heat at constant volume.

Once these constitutive relations are specified, the system becomes mathematically closed and can be solved for the primitive flow variables.

3.5 From Navier–Stokes to Euler equations

Once the governing equations have been closed through the introduction of appropriate constitutive relations, the conservation of momentum equation can be specialized to the case of a Newtonian fluid. Strictly speaking, the term *Navier–Stokes equation* refers to the balance of linear momentum in which the Newtonian constitutive relation for the viscous stress tensor has been substituted. By introducing the Newtonian stress tensor into the momentum equation, one obtains the Navier–Stokes equation in differential form:

$$\frac{\partial}{\partial t}(\rho\mathbf{V}) + \nabla\cdot(\rho\mathbf{V}\mathbf{V}) = -\nabla p + \nabla\cdot\left(\mu\left[\nabla\mathbf{V} + (\nabla\mathbf{V})^T\right] + \frac{2}{3}\mu(\nabla\cdot\mathbf{V})\mathbf{I}\right) + \rho\mathbf{f} \quad (3.31)$$

In modern literature, however, the expression *Navier–Stokes equations* is commonly used in a broader sense to denote the entire closed system governing a Newtonian, heat-conducting fluid, obtained after introducing both the Newtonian constitutive relation for the viscous stresses and Fourier's law for the heat flux, together with the continuity and energy equations.
In differential form, the full Navier–Stokes system reads:

$$\begin{cases} \dfrac{\partial\rho}{\partial t} + \nabla\cdot(\rho\mathbf{V}) = 0 \\ \dfrac{\partial}{\partial t}(\rho\mathbf{V}) + \nabla\cdot(\rho\mathbf{V}\mathbf{V}) = -\nabla p + \nabla\cdot\boldsymbol{\tau} + \rho\mathbf{f} \\ \dfrac{\partial}{\partial t}(\rho E) + \nabla\cdot(\rho E\mathbf{V}) = \nabla\cdot(\boldsymbol{\tau}\cdot\mathbf{V}) - \nabla\cdot\mathbf{q} - \nabla\cdot(p\mathbf{V}) + \rho\mathbf{f}\cdot\mathbf{V} \end{cases} \quad (3.32)$$

If viscous effects are neglected, as may occur at sufficiently high Reynolds numbers, and heat conduction effects are neglected, the system simplifies to:

$$\begin{cases} \dfrac{\partial\rho}{\partial t} + \nabla\cdot(\rho\mathbf{V}) = 0 \\ \dfrac{\partial}{\partial t}(\rho\mathbf{V}) + \nabla\cdot(\rho\mathbf{V}\mathbf{V}) = -\nabla p + \rho\mathbf{f} \\ \dfrac{\partial}{\partial t}(\rho E) + \nabla\cdot(\rho E\mathbf{V}) = -\nabla\cdot(p\mathbf{V}) + \rho\mathbf{f}\cdot\mathbf{V} \end{cases} \quad (3.33)$$

This reduced set of equations is known as the **Euler equations** and describes the motion of an inviscid and non-heat-conducting fluid. It is important to remark that the system of partial differential equations introduced above does not admit closed-form analytical solutions except for a limited number

of highly simplified configurations. For realistic engineering flows, characterized by complex geometries and nonlinear interactions, analytical treatment becomes intractable.
For this reason, the solution of the governing equations is most often obtained through numerical approaches, commonly referred to as Computational Fluid Dynamics (CFD). The discretization of the equations inevitably introduces numerical errors, including truncation, discretization, and round-off errors, which may affect the accuracy of the computed solution. Consequently, a solid physical understanding of the flow behavior is essential in order to critically assess the reliability and realism of numerical simulations. Finally, it should be emphasized that the governing equations have been presented in two mathematically equivalent forms: the integral (control-volume) form and the differential (local) form. The passage from the integral to the differential formulation relies on the application of the divergence theorem, which requires sufficient regularity of the fields involved, typically continuity and differentiability within the control volume. In the presence of discontinuities, such as shock waves, the differential form may no longer be strictly valid in the classical sense, and the integral formulation must be retained as the fundamental representation of the conservation laws.

Bibliography

[1] John D Anderson. *Fundamentals of aerodynamics*. McGraw-Hill Education, Columbus, OH, 6 edition, March 2016.

[2] John David Anderson. *Computational Fluid Dynamics*. McGraw-Hill series in mechanical engineering. McGraw-Hill Professional, New York, NY, February 1995.

[3] Frank M White. *Fluid Mechanics*. McGraw Hill Higher Education, Maidenhead, England, 7 edition, March 2010.

Chapter 4

Parametric Prediction of Nozzle Flow

The steady formulation of compressible flow problems may exhibit a mixed elliptic–hyperbolic character depending on the local Mach number. In subsonic regions, the equations behave elliptically and the solution is globally coupled, whereas in supersonic regions the system is hyperbolic and information propagates along characteristic directions. To avoid the numerical difficulties associated with this mixed-type behavior, modern CFD solvers typically consider the full time-dependent conservation laws and compute the steady solution as the asymptotic limit of a time-marching procedure. In this formulation, the Euler governing equations form a hyperbolic system, and the steady state is reached when temporal variations vanish. However, this approach introduces additional computational challenges. For explicit schemes, the allowable time step is restricted by the CFL condition and scales with the local mesh size. Consequently, the fine spatial resolution required to capture shock waves imposes very small time steps. Moreover, if the initial condition is far from the steady solution, a large number of iterations is required before convergence is achieved. Even when implicit schemes are employed to relax stability constraints, the cost per iteration increases significantly due to the need to solve large linear systems. As a result, the computation of steady compressible flows may become computationally demanding, particularly for high-resolution meshes and strongly nonlinear regimes. These considerations motivate the exploration of data-driven approaches not as full replacements of high-fidelity CFD solvers, but as predictive tools capable of providing an informed initial condition for the time-marching procedure. The objective of the present chapter is not to obtain a perfectly converged steady solution through the neural model alone. Instead, the proposed Graph Neural Network aims at predicting a flow field

that is sufficiently close to the final steady-state solution. Such an approximation can be used as an initial condition for the numerical solver, thereby significantly reducing the number of temporal iterations required to reach convergence and, consequently, the overall computational cost. In order to train and assess the proposed model, a database of CFD simulations was generated. The case study considered in this work consists of a converging–diverging nozzle, in which the outlet pressure was systematically varied. This configuration allows the flow to exhibit three distinct operating regimes: a fully subsonic regime, a transonic–supersonic regime with an internal normal shock in the divergent section, and a fully supersonic regime without internal shock formation. By exploring these different regimes, the dataset captures the rich nonlinear behavior characteristic of compressible flows, including shock formation and regime transitions.

4.1 Dataset Generation

4.1.1 Geometry and Mesh Generation

Geometry

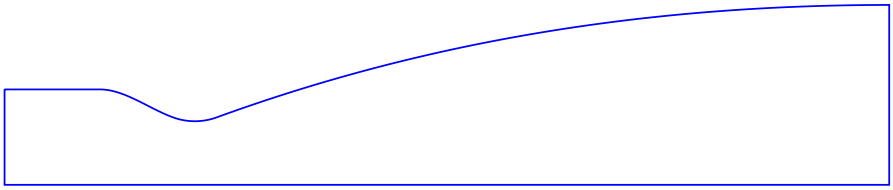


Figure 4.1: Nozzle geometry

The computational domain consists of a two-dimensional converging – diverging nozzle defined through a fully parametric geometry implemented in *Gmsh*. Due to the geometric symmetry of the configuration and the symmetric flow conditions considered in this study, only half of the nozzle is modeled. The centerline is treated as a symmetry boundary condition enforcing zero normal velocity at the boundary. This choice reduces the computational cost while preserving the physical consistency of the solution. The geometry is defined in terms of the following parameters:

$$R_t = 0.5\,\text{m} \quad \text{(throat radius)} \tag{4.1}$$

$$R_{ch} = 1.5\,R_t = 0.75\,\text{m} \quad \text{(chamber radius)} \tag{4.2}$$

$$\varepsilon = \frac{A_e}{A_t} = 8.0 \quad \text{(expansion ratio)} \tag{4.3}$$

The exit radius is obtained as:

$$R_e = \sqrt{\varepsilon}\,R_t = 1.414\,m \tag{4.4}$$

The converging and diverging sections are defined using prescribed wall angles:

$$\theta_t = 30^\circ \quad \text{(slope at throat)} \tag{4.5}$$

$$\theta_e = 0^\circ \quad \text{(exit slope)} \tag{4.6}$$

The chamber length is chosen :

$$L_{ch} = R_{ch} \tag{4.7}$$

The converging and diverging lengths are derived from conical reference geometries and subsequently rescaled to obtain a "bell-like" profile. The converging wall is constructed using a cubic Hermite spline connecting the chamber wall to the throat region. A cubic Hermite spline allows control over both the position and the slope at the endpoints. Given two points (x_0, y_0) and (x_1, y_1) with prescribed slopes m_0 and m_1, the spline is defined as:

$$y(s) = h_{00}(s)\, y_0 + h_{10}(s)\, (x_1 - x_0)\, m_0 + h_{01}(s)\, y_1 + h_{11}(s)\, (x_1 - x_0)\, m_1 \tag{4.8}$$

where $s \in [0, 1]$ and the Hermite basis functions are:

$$\begin{aligned} h_{00}(s) &= 2s^3 - 3s^2 + 1 \\ h_{10}(s) &= s^3 - 2s^2 + s \\ h_{01}(s) &= -2s^3 + 3s^2 \\ h_{11}(s) &= s^3 - s^2 \end{aligned} \tag{4.9}$$

To ensure a smooth transition between the converging and diverging sections, a circular fillet is introduced at the throat. The fillet guarantees curvature regularity and avoids geometric discontinuities that could negatively affect the numerical solution. The fillet radius is prescribed as

$$R_{\text{fil}} = R_t. \tag{4.10}$$

For the convergin section the geometric constraints for the spline are defined as follows:

- The starting point coincides with the end of the straight chamber wall. At this location the wall slope is zero ($m_0 = 0$).
- The endpoint corresponds to the tangency point with the throat fillet. The slope at this point is prescribed as $m_1 = -\tan(\theta_t)$, ensuring slope continuity with the circular arc.

This construction guarantees C^1 continuity between the chamber wall, the converging spline, and the throat region. The geometric construction of the converging section is illustrated in Figure 4.2.

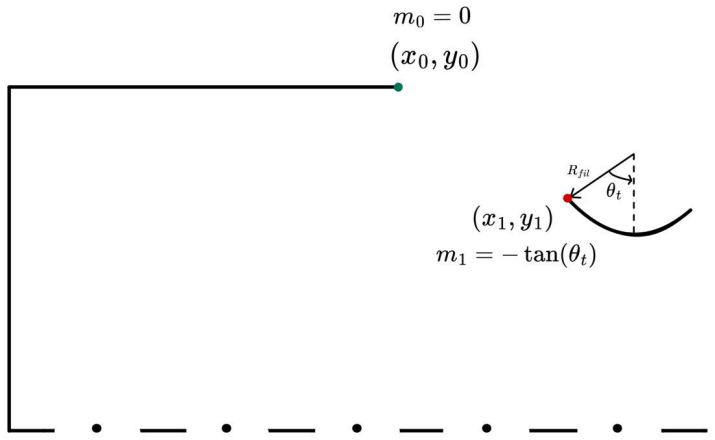


Figure 4.2: Geometric construction of the converging section

The diverging section is also constructed using a cubic Hermite spline but in this case the gemotric costrain are apllied as follows:

- The starting point coincides with the tangency point between the circular throat fillet and the divergent profile. At this location the wall slope is $m_0 = \tan(\theta_t)$.
- The endpoint corresponds to the nozzle exit, where the radius equals R_e and the wall inclination is prescribed as $m_1 = \tan(\theta_e)$.

This construction guarantees C^1 continuity between the throat region and the divergent section, producing a smooth bell-shaped contour with controlled curvature and gradual area expansion.

Mesh

The computational domain is discretized using an unstructured mesh composed of triangular elements generated through the Delaunay algorithm. No a priori local refinement was applied to specifically capture potential shock waves, as their location depends on the outlet pressure and varies across the parametric study. Consequently, the mesh resolution was selected as a trade-off between adequately resolving shock structures and limiting the overall computational cost. This choice ensures sufficient accuracy in representing the main flow features while maintaining reasonable simulation times throughout the entire range of operating conditions.

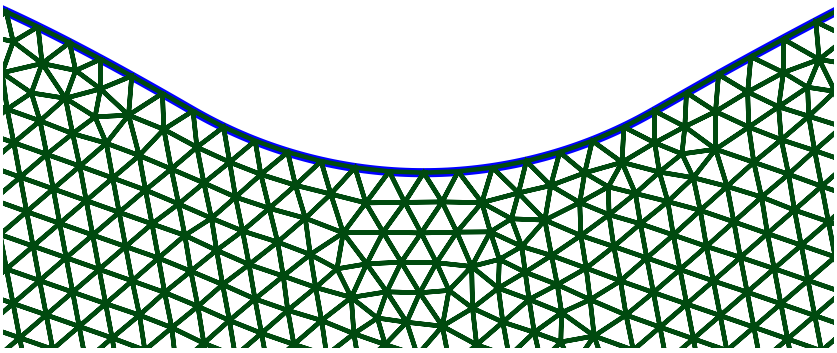


Figure 4.3: Nozzle mesh

4.1.2 Numerical Discretisation

The flow is modeled by the two-dimensional compressible Euler equations in conservative form, neglecting body-force terms:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} = \mathbf{0} \tag{4.11}$$

where the vector of conservative variables is defined as:

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix} \tag{4.12}$$

and the flux vectors in the x and y directions are respectively:

$$\mathbf{F}(\mathbf{U}) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (\rho E + p)u \end{pmatrix}, \qquad \mathbf{G}(\mathbf{U}) = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ (\rho E + p)v \end{pmatrix} \tag{4.13}$$

A density-based finite volume formulation is adopted. For each control volume V_i, the semi-discrete form of the governing equations can be written as

$$\frac{d\mathbf{U}_i}{dt} = -\frac{1}{|V_i|} \sum_{f \in \partial V_i} \hat{\mathbf{F}}_f \, |S_f| \tag{4.14}$$

where $\hat{\mathbf{F}}_f$ represents the numerical flux across face f and $|S_f|$ denotes the face area. The numerical flux at each interface is computed using the local Lax–Friedrichs scheme:

$$\hat{\mathbf{F}}(\mathbf{U}_L, \mathbf{U}_R; \mathbf{n}) = \frac{1}{2}\left[\mathbf{F}_n(\mathbf{U}_L) + \mathbf{F}_n(\mathbf{U}_R)\right] - \frac{1}{2}\alpha\left(\mathbf{U}_R - \mathbf{U}_L\right) \tag{4.15}$$

where the normal physical flux is defined as:

$$\mathbf{F}_n(\mathbf{U}) = \mathbf{F}(\mathbf{U})\, n_x + \mathbf{G}(\mathbf{U})\, n_y \tag{4.16}$$

with $\mathbf{n} = (n_x, n_y)$ denoting the outward unit normal vector to the face. The dissipation coefficient α is defined as the maximum local characteristic speed:

$$\alpha = \max\left(|u_{n,L}| + c_L,\ |u_{n,R}| + c_R\right) \tag{4.17}$$

where:

$$u_n = \mathbf{u} \cdot \mathbf{n}, \qquad c = \sqrt{\gamma \frac{p}{\rho}} \tag{4.18}$$

This formulation introduces a local, solution-dependent dissipation proportional to the maximum characteristic speed at the interface, providing robustness in the presence of strong gradients and shock waves preventing non-physical oscillations near shock waves. Eventually the semi-discrete system is advanced in time using a first-order explicit Euler scheme. Denoting by $\mathbf{R}_i(\mathbf{U})$ the spatial residual associated with control volume V_i, the time integration reads

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n + \Delta t\, \mathbf{R}_i^n \tag{4.19}$$

The global time step Δt is determined from the CFL stability constraint. Defining a local Courant number as

$$\mathrm{Co}_i = \frac{\left(|\mathbf{u}_i| + a_i\right)\Delta t}{\sqrt{A_i}}, \tag{4.20}$$

the time step is chosen such that $\mathrm{Co}_i \leq \mathrm{CFL}$ in every cell. This yields

$$\Delta t = \mathrm{CFL}\ \min_i \left(\frac{\sqrt{A_i}}{|\mathbf{u}_i| + a_i}\right), \tag{4.21}$$

where CFL = 0.3 in the present simulations. In the above expression, A_i represents the area of the i-th cell, $|\mathbf{u}_i|$ the velocity magnitude, and a_i the local speed of sound. The use of a global time step determined by the most restrictive local condition ensures numerical stability of the explicit scheme. However, this choice may significantly limit the allowable time step in the presence of small cells or regions characterized by high local wave speeds.

In compressible flows, the characteristic velocity $|\mathbf{u}| + a$ can become large, particularly in regions of strong acceleration or near shock waves. As a consequence, the CFL constraint may impose very small time steps. Since the steady solution is obtained as the asymptotic limit of the time-marching procedure, a large number of iterations may be required when the initial condition is far from the steady state. This computational cost motivates the exploration of data-driven approaches capable of providing an informed initial condition closer to the final steady solution, thereby reducing the number of temporal iterations required for convergence.

4.1.3 Parametric CFD Simulations and Data Pre – Procssing

A total of 28 simulations were performed. In each simulation, the total pressure inside the chamber was kept constant, while the ratio between the chamber total pressure and the external static pressure was progressively varied. More precisely, the parameter:

$$\Pi = \frac{p_{\text{ext}}}{p_{0,\text{chamber}}} \tag{4.22}$$

was varied from 0.1 to 0.98, thus generating a parametric dataset in which the only varying quantity is the pressure ratio. This procedure allows isolating the influence of the external pressure conditions on the resulting flow field. The numerical solver developed for this study adopts a cell-centered finite volume formulation; therefore, the primary physical quantities are defined at the center of each control volume. In order to construct a graph representation suitable for vertex-based message passing, the cell-centered quantities were interpolated onto the mesh vertices. The interpolation was performed using a Least Mean Squared Error (LMSE) reconstruction procedure. For each vertex, the vertex value was locally reconstructed from the surrounding cell-centered values by minimizing a quadratic error functional, ensuring a smooth and physically consistent projection of the solution from control volume centers to vertices. This step preserves the spatial coherence of the numerical solution while transforming the dataset into a nodal representation compatible with Graph Neural Networks.

4.2 Graph-based representation and Message – Passing architecture

4.2.1 Graph construction

For each simulation, a graph $G = (V, E)$ was constructed, where:

- V represents the set of mesh vertices
- E represents the set of edges defined according to mesh connectivity

Each simulation corresponds to one graph sample within the dataset. Each node $v_i \in V$ is associated with the following feature vector:

$$\mathbf{x}_i = [x,\ y,\ \text{sect\_h},\ \text{inlet},\ \text{outlet},\ \text{wall},\ \text{fluid}] \tag{4.23}$$

where:

- x, y are the spatial coordinates
- sect_h represents the local cross-sectional size at the point location
- inlet, outlet, wall, fluid are binary indicators encoding the boundary condition type

This representation embeds both geometric information and boundary condition encoding directly into the nodal feature space. In addition to the nodal attributes, geometric relationships between neighboring nodes were explicitly encoded at the edge level. Given an edge connecting node i (source) and node j (destination), the corresponding edge features were constructed using relative geometric quantities derived from the nodal features. The relative displacement vector was defined as

$$\Delta\mathbf{x}_{ij} = \begin{bmatrix} x_j - x_i \\ y_j - y_i \end{bmatrix} \tag{4.24}$$

with associated Euclidean distance:

$$d_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \tag{4.25}$$

In addition to the relative spatial displacement, a geometric variation term was included to provide the model with information about local changes in the cross-sectional size. In particular, the variation of the sectional parameter between two connected nodes was defined as

$$\Delta\text{sect\_h}_{ij} = \text{sect\_h}_j - \text{sect\_h}_i \tag{4.26}$$

This quantity was introduced to explicitly inform the model about how rapidly the cross-sectional dimension changes between neighboring nodes. By encoding this local geometric variation, the network can better capture expansion or contraction effects along the flow direction, which are directly related to the governing fluid-dynamic behavior. The complete edge feature vector was therefore defined as:

$$\mathbf{e}_{ij} = [\Delta x_{ij}, \Delta y_{ij}, d_{ij}, \Delta\text{sect\_h}_{ij}] \tag{4.27}$$

This formulation allows the Graph Neural Network to incorporate relative spatial and geometric information during message passing, consistently with the local interaction structure of the underlying conservation laws.

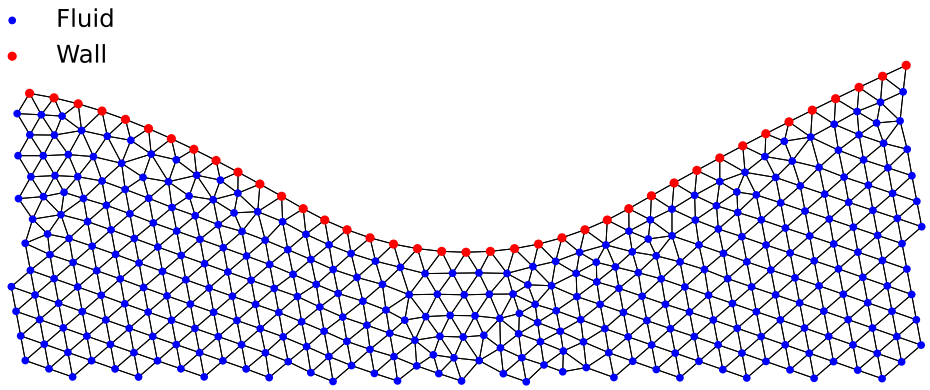


Figure 4.4: Graph representation of the Nozzle computational domain

4.2.2 Design of the Message Passing Architecture

The proposed architecture follows an **Encoder–Message Passing–Decoder** structure, specifically designed to operate on geometry-augmented graph representations with global conditioning.

Encoding stage

The initial node features $\mathbf{x}_i \in \mathbb{R}^7$ and edge features $\mathbf{e}_{ij} \in \mathbb{R}^4$ are first projected into a latent space of dimension 128 through independent multi-layer

perceptrons (MLPs). Denoting the encoded node and edge representations as $\mathbf{h}_i^{(0)}$ and $\tilde{\mathbf{e}}_{ij}$, the encoding step can be written as:

$$\mathbf{h}_i^{(0)} = \phi_{\text{enc}}(\mathbf{x}_i), \qquad \tilde{\mathbf{e}}_{ij} = \psi_{\text{enc}}(\mathbf{e}_{ij}) \tag{4.28}$$

where ϕ_{enc} and ψ_{enc} are learnable nonlinear mappings. Before the propagation process, the latent node embeddings are modulated using the global pressure ratio parameter Π, encoding the pressure ratio associated with each simulation. Two learnable transformations are introduced:

$$\boldsymbol{\gamma} = \tanh(W_\gamma \, \Pi), \qquad \boldsymbol{\beta} = W_\beta \, \Pi \tag{4.29}$$

and the encoded node features are updated as

$$\mathbf{h}_i^{(0)} = \boldsymbol{\gamma} \odot \mathbf{h}_i^{(0)} + \boldsymbol{\beta} \tag{4.30}$$

where $\odot$ denotes element-wise multiplication. This mechanism enables the model to adapt its internal representation to different pressure ratios while preserving a shared geometric structure.

Message passing stage

Let $\mathbf{h}_i^{(k)} \in \mathbb{R}^H$ denote the latent representation of node i at layer k. For each edge (i, j), a message is computed as

$$\mathbf{m}_{ij}^{(k)} = \phi_m \left(\mathbf{h}_i^{(k)} \parallel \mathbf{h}_j^{(k)} \parallel \tilde{\mathbf{e}}_{ij} \right) \tag{4.31}$$

where $\parallel$ denotes vector concatenation and $\phi_m(\cdot)$ is a learnable MLP. Incoming messages are aggregated using a mean operator

$$\mathbf{m}_i^{(k)} = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij}^{(k)} \tag{4.32}$$

and subsequently refined through another MLP $\phi_u(\cdot)$. A residual update rule is applied as follows:

$$\mathbf{h}_i^{(k+1)} = \mathbf{h}_i^{(k)} + \phi_u \left(\mathbf{m}_i^{(k)} \right) \tag{4.33}$$

which improves optimization stability, facilitates deeper architectures, and mitigates oversmoothing effects.

Decoding stage

After 12 message passing layers, the final latent representation $\mathbf{h}_i^{(12)}$ is mapped to the target physical quantities through a decoding MLP:

$$\hat{\mathbf{y}}_i = \phi_{\text{dec}}\left(\mathbf{h}_i^{(L)}\right) \tag{4.34}$$

producing the predicted flow variables at each node. Overall, the resulting model can be interpreted as a globally conditioned residual MPNN operating on geometry-enriched graph features, consistent with the local interaction structure of the governing conservation laws.

4.3 Training and Validation Procedure

The training process was performed using a supervised learning strategy based node-level targets. The available graph dataset was randomly divided into a training and a validation subset, using 80% of the graphs for training and the remaining 20% for validation. This split ensured that model performance was evaluated on unseen flow configurations corresponding to different pressure ratios. The model parameters were optimized with the Adam optimizer using a learning rate of 10^{-3}. A learning-rate scheduler based on `ReduceLROnPlateau` was employed, monitoring the validation loss and reducing the learning rate by a factor 0.8 after 30 epochs without improvement. Training was carried out for up to 2500 epochs, while model weights selection was performed by retaining the network weights that achieved the lowest validation error. During optimization, the loss function was computed in a normalized space to improve numerical conditioning. Let $\hat{\mathbf{y}}_i$ and $\mathbf{y}_i$ denote the predicted and reference targets at node i, expressed in normalized form. The training loss was defined as a node-wise mean-squared error (MSE), averaged over the output channels and subsequently averaged over all nodes in the batch:

$$\mathcal{L}_{\text{norm}} = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{1}{C} \sum_{c=1}^{C} (\hat{y}_{i,c} - y_{i,c})^2 \right), \tag{4.35}$$

where N denotes the total number of nodes in the batch and C represents the number of predicted physical quantities. In order to prevent unstable updates, gradient clipping was applied at each iteration, enforcing a maximum gradient norm equal to 1.0.

For monitoring and reporting purposes, the loss was additionally evaluated in physical units by denormalizing predictions and targets through the stored mean and standard deviation vectors, i.e. $\hat{\mathbf{y}}^{\text{phys}} = \hat{\mathbf{y}} \odot \boldsymbol{\sigma}_y + \boldsymbol{\mu}_y$ and $\mathbf{y}^{\text{phys}} = \mathbf{y} \odot \boldsymbol{\sigma}_y + \boldsymbol{\mu}_y$. The validation physical loss was used as the reference metric for learning-rate scheduling and for selecting the best-performing model checkpoint.

4.4 Results

This section presents the evaluation of the proposed graph-based model across different flow regimes. The analysis aims to assess both the qualitative and quantitative agreement between the GNN predictions and the reference CFD solutions. To provide a physically meaningful interpretation of the model performance, the results are organized according to three representative distinct nozzle operating regime: a fully subsonic regime, a transonic–supersonic regime with an internal normal shock in the divergent section, and a fully supersonic regime without internal shock formation. These regimes correspond to different values of the pressure ratio parameter $\mathbf{\Pi}$ and are characterized by distinct physical behaviors, ranging from smooth Mach distributions to the presence of internal shock waves. All the graph samples presented in this section belong to the validation subset and were not used during training. Therefore, the comparisons shown below evaluate the generalization capability of the network on previously unseen flow configurations. For each regime, a comparison between the target CFD field and the predicted field is presented, together with the corresponding error distribution. Additional quantitative metrics are used to evaluate the global predictive performance of the model across the entire parametric range.

Subsonic Regime

In the fully subsonic regime, the imposed back pressure is sufficiently high to prevent choking at the throat, and the flow remains subsonic throughout the entire nozzle. As a consequence, the Mach number field exhibits a smooth acceleration in the converging section up to a subsonic peak in the vicinity of the throat, followed by a gradual deceleration along the diverging part, with no discontinuities or shock-related features. Figures 4.5.(a) and 4.5.(b) show the comparison between the CFD reference Mach field and the corresponding GNN prediction for a representative subsonic configuration. The global structure is accurately reproduced: the location of the peak near the throat and the progressive reduction of Mach number downstream are well captured, and the overall spatial distribution remains consistent with the reference solution. A more quantitative assessment is provided by the centerline profile reported in Figure 4.6. The predicted curve closely follows the CFD solution, correctly reproducing the maximum Mach level near the throat and the subsequent monotonic decay in the diverging region. Minor discrepancies appear downstream of the throat, where the GNN shows a slightly smoother recovery toward lower Mach values, but the overall trend and curvature remain in good agreement. The spatial error distribution (Figure 4.5.(c)) confirms

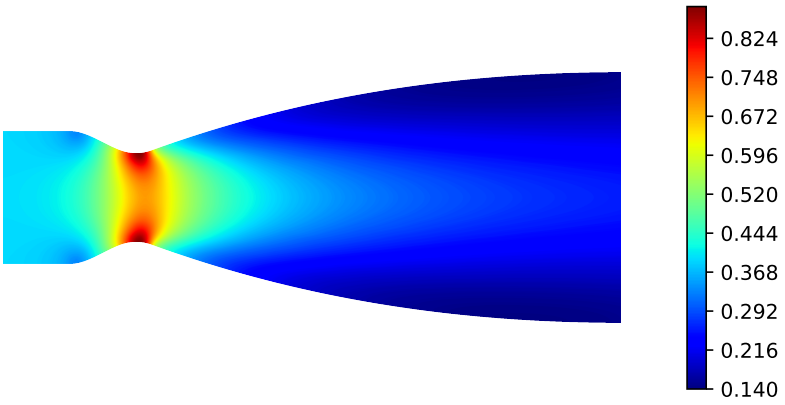
that, in the absence of discontinuities, the deviations remain limited and mainly concentrated in regions of stronger gradients, particularly around the throat, while the remainder of the domain exhibits uniformly low relative error. The relative Mach error is defined as:

$$\varepsilon = \frac{|M_{\mathrm{CFD}} - M_{\mathrm{GNN}}|}{\max\left(M_{\mathrm{CFD}}\right)}. \tag{4.36}$$

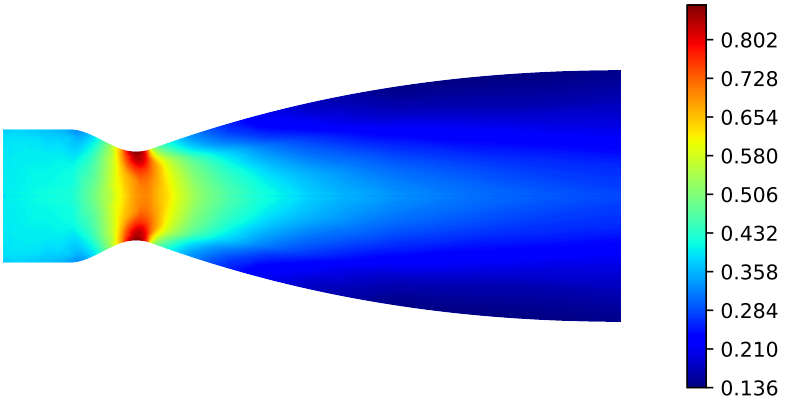
where the normalization is performed with respect to the maximum Mach number of the CFD solution. From a quantitative perspective, the domain-averaged relative error is 0.96 %, while the maximum relative error over the full field reaches about 11.22 %. For completeness, the predictive capability of the proposed architecture is further assessed by analyzing the non-dimensional static pressure distribution in the subsonic regime. Consistently with the smooth behavior observed for the Mach field, the pressure solution exhibits a continuous evolution along the nozzle, characterized by a pressure drop in the vicinity of the throat followed by a gradual recovery in the diverging section. Figures 4.7.(a) and 4.7.(b) compare the CFD reference solution and the corresponding GNN prediction over the entire computational domain. The GNN accurately reproduces the global structure of the pressure field and preserves the correct coupling between velocity and thermodynamic variables in this smooth-flow configuration. A more quantitative comparison is provided by the centerline profile reported in Figure 4.8. Similarly to the Mach distribution, the predicted pressure curve closely follows the CFD reference along the nozzle. However, unlike the Mach case, the agreement in the downstream recovery region is slightly improved, while a somewhat larger discrepancy is observed around the peak (in the vicinity of the throat), where the pressure gradients are strongest. The spatial error distribution (Figure 4.7.(c)) reflects the same qualitative pattern observed for Mach, with localized peaks mainly confined to the throat neighborhood and low error elsewhere. The relative pressure error is evaluated over the entire computational domain according to:

$$\varepsilon_p = \frac{|p_{\mathrm{CFD}} - p_{\mathrm{GNN}}|}{\max\left(p_{\mathrm{CFD}}\right)}. \tag{4.37}$$

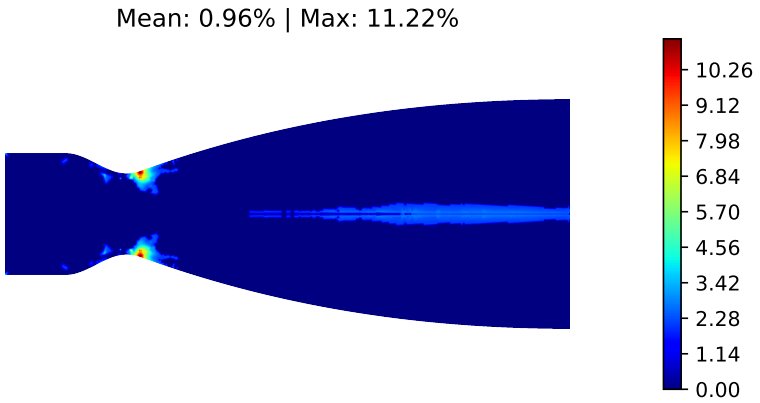
Overall, the global agreement remains satisfactory. From a quantitative perspective, the domain-averaged relative error is 0.35 %, while the maximum relative error over the full field reaches about 7.00 %.



(a) CFD Mach target



(b) GNN Mach prediction



(c) Relative error map

Figure 4.5: Comparison of Mach number distribution for $\Pi = 0.93$

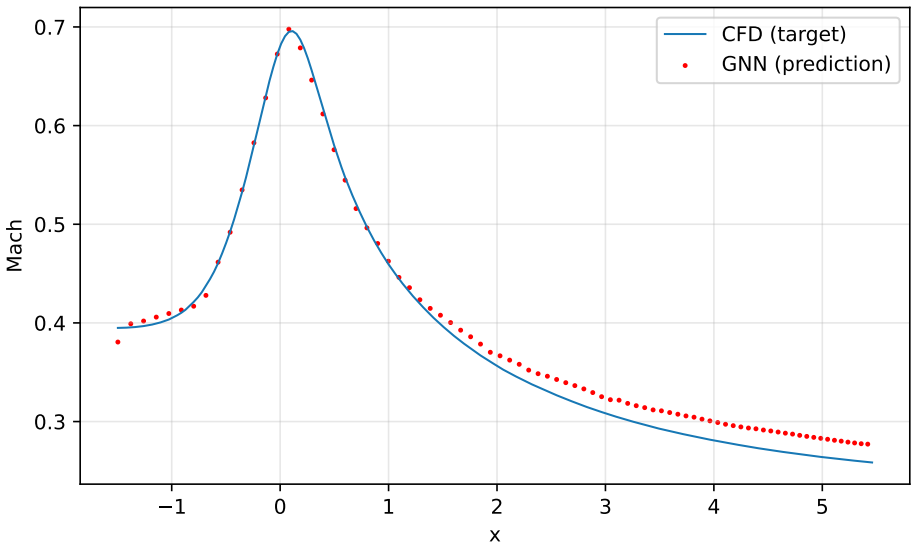
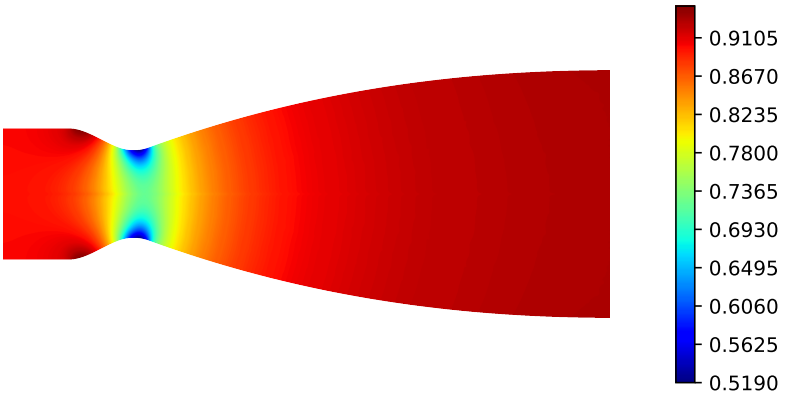
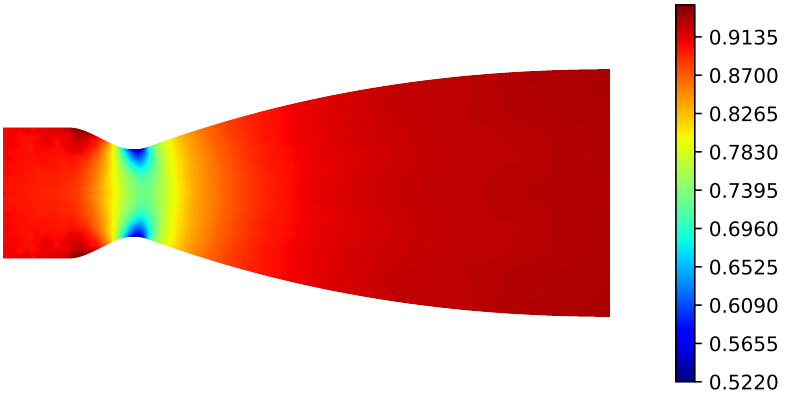


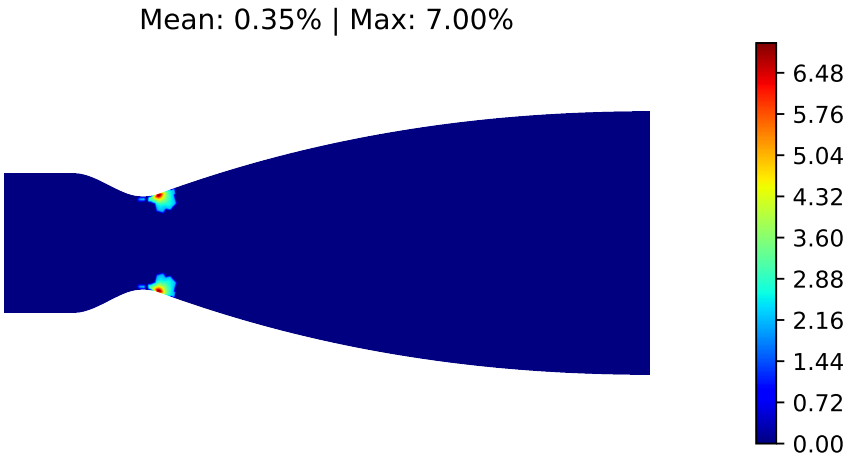
Figure 4.6: Centerline Mach profile



(a) CFD non-dimensional Pressure target



(b) GNN non-dimensional Pressure prediction



(c) Relative error map

Figure 4.7: Comparison of non-dimensional pressure distribution for $\Pi = 0.93$

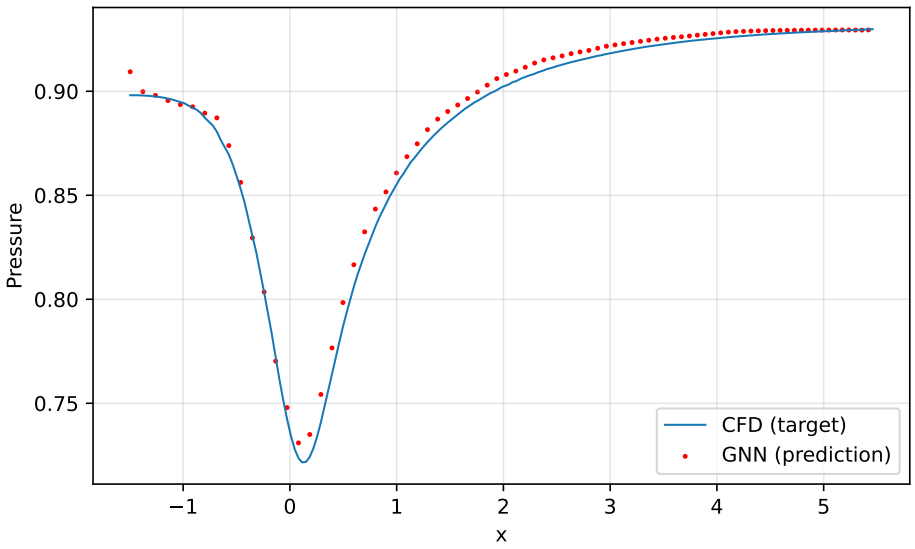


Figure 4.8: Centerline non-dimensional pressure profile

Subsonic–Supersonic Regime

In the subsonic–supersonic regime, the flow reaches supersonic conditions downstream of the throat but undergoes an internal normal shock within the diverging section of the nozzle. This configuration corresponds to pressure ratios $\mathbf{\Pi}$ sufficiently low to induce choking at the throat, while the imposed back pressure is still high enough to force a shock adjustment inside the nozzle. As a consequence, the Mach number distribution is characterized by a rapid acceleration to supersonic values followed by an abrupt discontinuity associated with the shock, and a subsequent subsonic recovery toward the exit. Figures 4.9.(a) and 4.9.(b) compare the reference CFD Mach field with the corresponding GNN prediction for a representative overexpanded configuration. The two-dimensional distributions clearly highlight the presence of a strong normal shock in the diverging section. The GNN successfully reproduces the global acceleration pattern and the overall shock-induced transition, yielding a flow topology consistent with the reference solution: an upstream supersonic core followed by a downstream subsonic region. Minor transverse (radial) oscillations are visible in the predicted field, but their amplitude remains limited and does not alter the macroscopic structure. The centerline profile reported in Figure 4.10.(a) enables a pointwise comparison along the nozzle axis. At the scale of the full-domain plot, the predicted curve closely follows the CFD profile and reproduces the main jump across the shock. However, a magnified view of the shock region (Figure 4.10.(b)) reveals two typical local effects: the discontinuity is predicted with a small streamwise shift and a slightly more gradual transition, i.e. a smeared shock thickness compared to the CFD reference. This indicates that, while the shock is captured in a qualitatively correct manner, its exact position and sharpness are the primary sources of discrepancy at high magnification. This behavior is also reflected by the spatial error distribution (Figure 4.9.(c)), where the largest deviations are confined to a narrow band around the discontinuity, i.e. the region characterized by the steepest gradients. Such localized errors are consistent with the intrinsic difficulty of representing discontinuous solutions using continuous function approximators, which tend to distribute the jump over a finite spatial extent. The relative error is defined as:

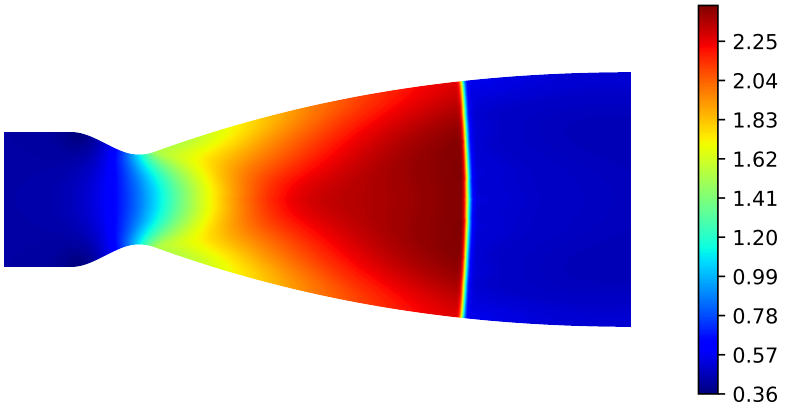
$$\varepsilon = \frac{|M_{\mathrm{CFD}} - M_{\mathrm{GNN}}|}{\max\left(M_{\mathrm{CFD}}\right)}. \tag{4.38}$$

The domain-averaged relative error is 0.48 %, while the maximum relative error reaches about 26.00 %. Despite the presence of a strong shock, the global agreement remains satisfactory, and the errors remain strongly lo-

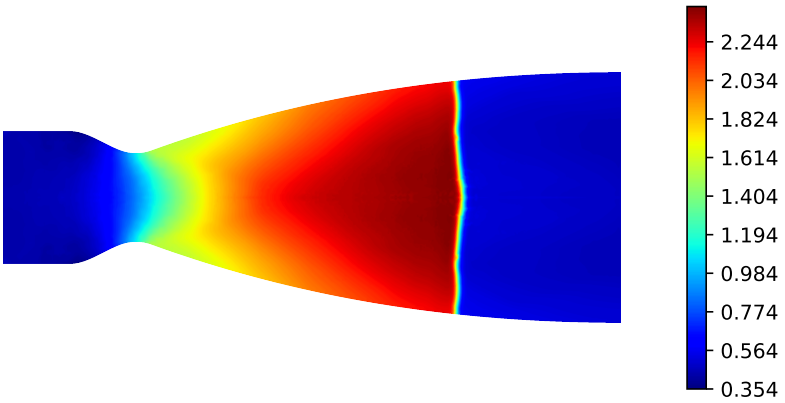
calized around the discontinuity, confirming that the proposed GNN architecture can capture complex nonlinear phenomena such as shock-induced flow transitions, with the main residual limitations concentrated on the precise shock alignment and sharpness. For completeness, the predictive capability of the proposed architecture is further assessed by analyzing the non-dimensional static pressure distribution. Figures 4.11.(a) and 4.11.(b) compare the CFD reference solution with the corresponding GNN prediction for the non-dimensional static pressure over the entire computational domain. The global structure of the pressure field is accurately reproduced, including the pronounced pressure drop through the accelerating section, the low-pressure supersonic core, and the pressure rise associated with the internal normal shock in the diverging region. To provide a more quantitative comparison, the pressure evolution along the symmetry axis is reported in Figure 4.12.(a). The centerline profile enables a direct pointwise comparison, particularly in the vicinity of the shock, where the pressure gradient is extremely steep. As shown in Figure 4.12.(a), the predicted curve closely follows the reference CFD profile throughout the nozzle, reproducing the monotonic pressure decrease in the supersonic region and the overall pressure jump across the shock. Nevertheless, consistently with the behavior previously observed for the Mach number (Figure 4.10.(b)), a magnified view of the shock region (Figure 4.12.(b)) reveals a small streamwise mismatch in the discontinuity position, together with a slightly more gradual transition compared with the CFD solution. This confirms that the main residual limitations of the surrogate are concentrated on the sharpest gradients: the discontinuity is captured in a qualitatively correct manner and with an accurate jump magnitude, while the precise shock alignment and sharpness are slightly degraded when examined at high magnification. Consistently with this observation and in agreement with the Mach-error patterns, the spatial error distribution (Figure 4.11.(c)) highlights localized discrepancies confined to a narrow band around the discontinuity, where the pressure gradient is largest. As in the Mach case, such localized errors are expected when approximating sharp transitions with continuous neural network models and do not compromise the overall predictive accuracy. The relative pressure error is evaluated over the entire computational domain according to

$$\varepsilon_p = \frac{|p_{\mathrm{CFD}} - p_{\mathrm{GNN}}|}{\max\left(p_{\mathrm{CFD}}\right)}. \tag{4.39}$$

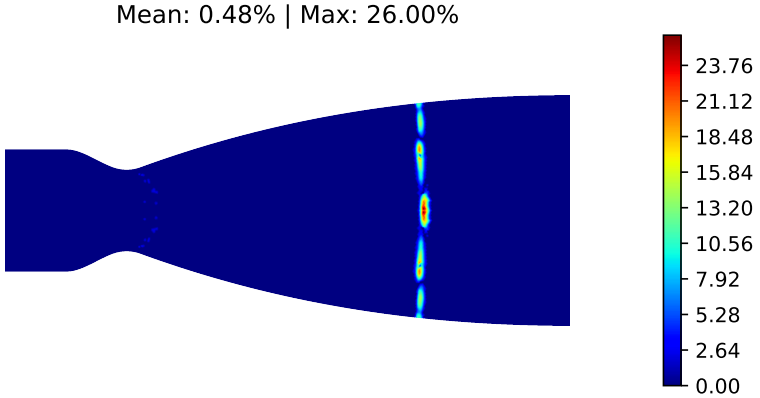
Despite the presence of a strong compressive discontinuity, the global agreement remains satisfactory. From a quantitative perspective, the domain-averaged relative error is 0.45 %, while the maximum relative error over the full field reaches about 11.00 %.



(a) CFD Mach target

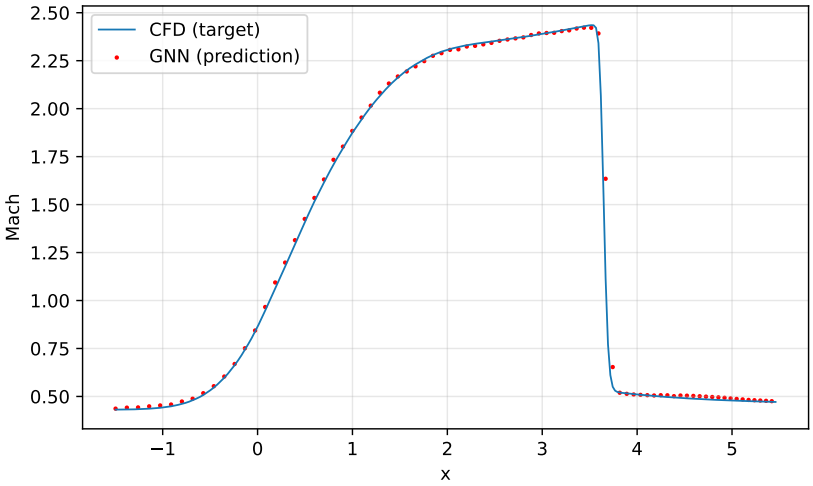


(b) GNN Mach prediction

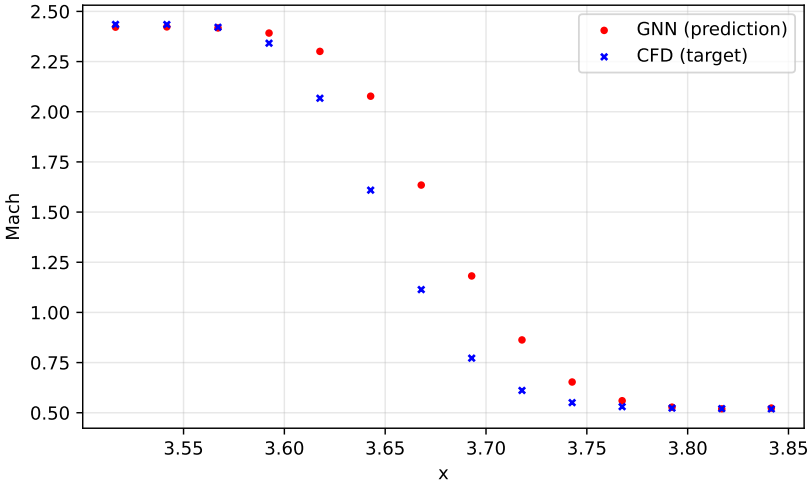


(c) Relative error map

Figure 4.9: Comparison of Mach number distribution for $\Pi = 0.42$

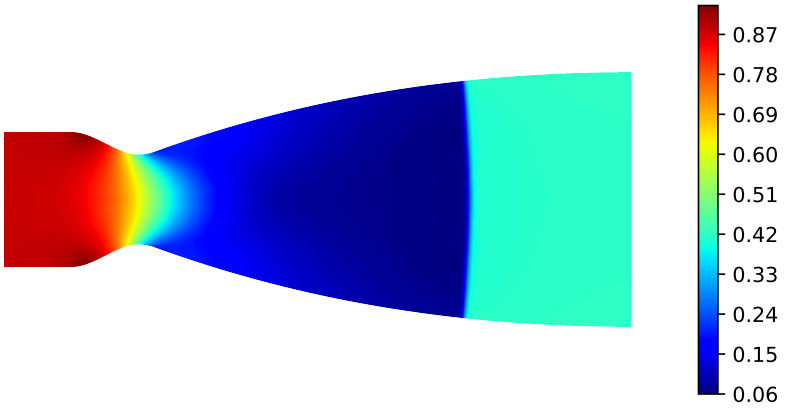


(a) Centerline Mach profile

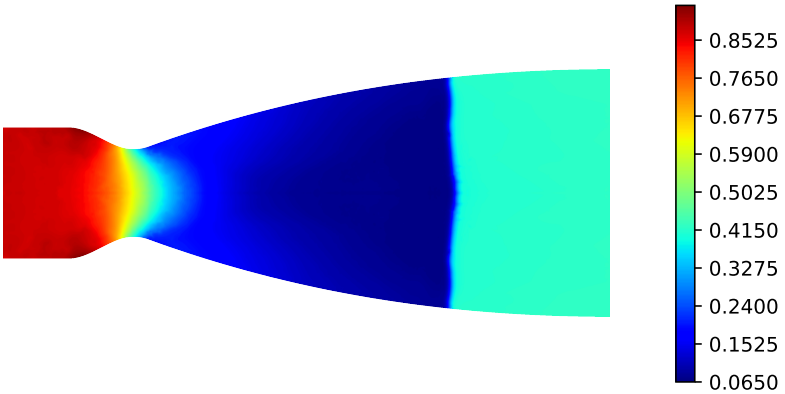


(b) Shock-region zoom

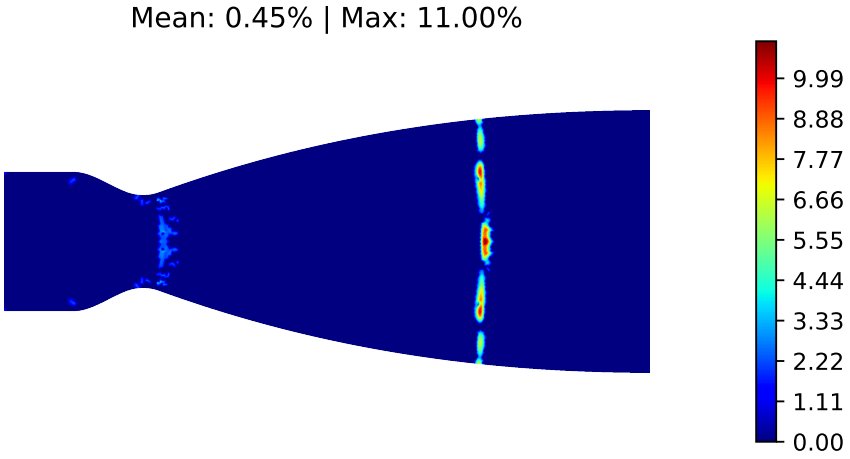
Figure 4.10: Centerline Mach: full profile and shock-region zoom



(a) CFD non-dimensional Pressure target

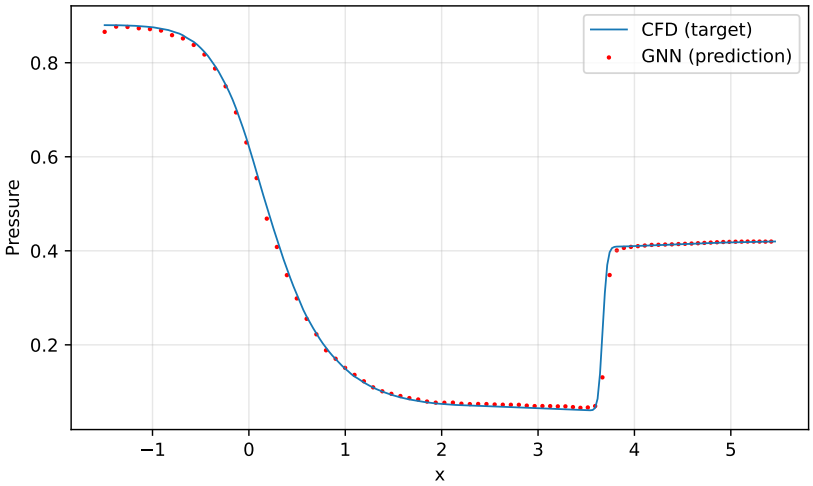


(b) GNN non-dimensional Pressure prediction

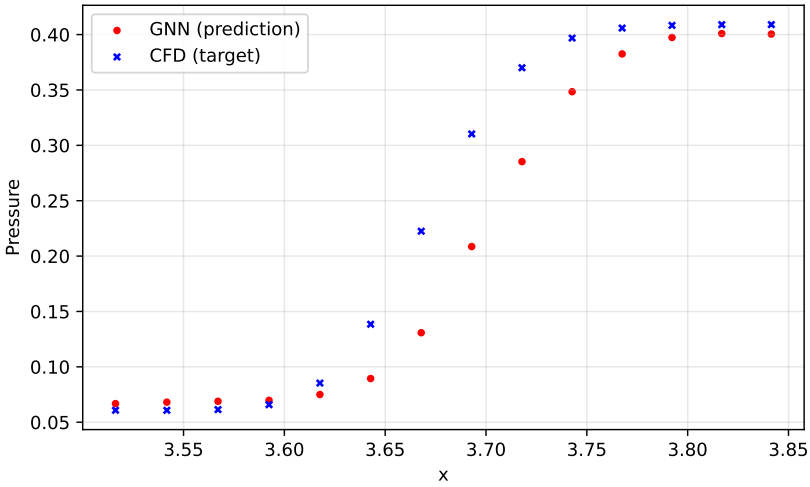


(c) Relative error map

Figure 4.11: Comparison of non-dimensional pressure distribution for $\Pi = 0.42$



(a) Centerline Mach profile



(b) Shock-region zoom

Figure 4.12: Centerline non-dimensional pressure: full profile and shock-region zoom

Supersonic Regime

In the fully supersonic regime, the flow becomes choked at the throat and continues to accelerate smoothly throughout the diverging section, without the formation of an internal shock. This configuration corresponds to sufficiently low pressure ratios $\mathbf{\Pi}$, such that the imposed back pressure allows a fully expanded supersonic flow at the nozzle exit. As a consequence, the Mach number distribution is characterized by a monotonic and continuous acceleration from subsonic conditions upstream of the throat to increasingly higher supersonic values downstream, without any discontinuity. Figures 4.13.(a) and 4.13.(b) show the comparison between the reference CFD Mach number distribution and the corresponding GNN prediction for a representative fully supersonic configuration. The two-dimensional fields exhibit a smooth and continuous acceleration pattern along the nozzle, with no abrupt transitions. The GNN prediction accurately reproduces the global acceleration trend, the spatial distribution of the supersonic core, and the progressive increase of Mach number toward the exit. The centerline profile, reported in Figure 4.14, provides a direct pointwise comparison between the CFD solution and the GNN prediction. As shown in Figure 4.14, the predicted curve closely overlaps the reference solution along the entire nozzle length. The monotonic acceleration and the curvature of the profile are accurately captured, confirming the capability of the model to approximate smooth compressible flow fields with high fidelity. The spatial error distribution (Figure 4.13.(c)) further confirms this behavior. The relative error remains uniformly low throughout the domain, with a mean value of 0.34 % and a maximum value of 3.40 %. Unlike the subsonic–supersonic regime, no localized peaks associated with discontinuities are observed. The remaining discrepancies are mainly concentrated in regions of higher gradient near the throat, where the acceleration is most pronounced. The relative error is defined as:

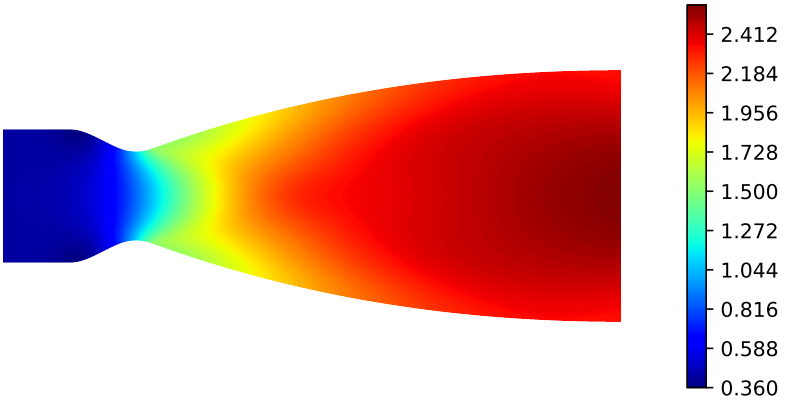
$$\varepsilon = \frac{|M_{\mathrm{CFD}} - M_{\mathrm{GNN}}|}{\max\left(M_{\mathrm{CFD}}\right)} \tag{4.40}$$

where the normalization is performed with respect to the maximum Mach number of the CFD solution. From a quantitative perspective, the significantly lower maximum error compared to the shock-containing configuration highlights the robustness of the proposed GNN architecture in modeling smooth nonlinear compressible flow regimes. For completeness, the predictive capability of the proposed architecture is further assessed by analyzing the non-dimensional static pressure distribution in the fully supersonic regime. As already observed for the Mach number field, the pressure solution exhibits a smooth and monotonic behavior throughout the diverging section, with no

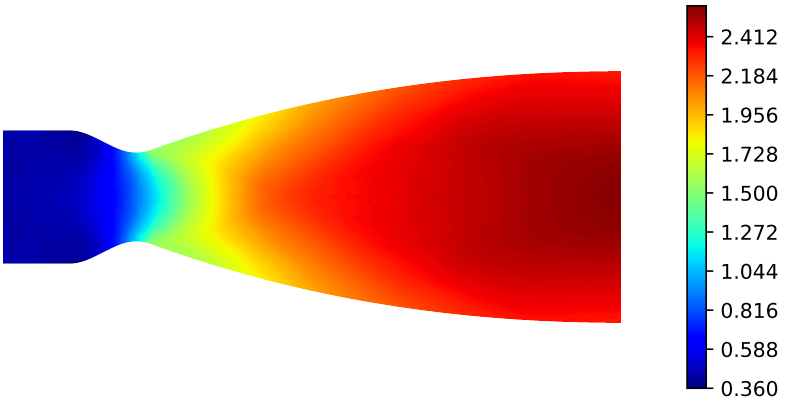
internal shock or discontinuity. Figures 4.15.(a) and 4.15.(b) compare the CFD reference solution and the corresponding GNN prediction over the entire computational domain. Consistently with the Mach results, the GNN accurately reproduces the global structure of the pressure field, capturing both the strong pressure drop downstream of the throat and the gradual asymptotic trend toward the exit. The agreement between the two-dimensional fields confirms that the network preserves the correct coupling between velocity and thermodynamic variables in the fully supersonic regime. A more quantitative comparison is provided by the centerline profile reported in Figure 4.16. Similarly to the Mach distribution, the predicted pressure curve closely overlaps the CFD reference along the entire nozzle length, accurately reproducing the monotonic decrease and the curvature of the solution. The spatial error distribution (Figure 4.15.(c)) further reflects the same behavior observed for the Mach field. The relative pressure error remains uniformly low across the domain, with a mean value of 0.40 % and a maximum value of 5.13 %. In the absence of discontinuities, no localized error amplification is detected, and the remaining discrepancies are mainly concentrated near the throat, where the gradients are strongest. The relative pressure error is evaluated over the entire computational domain according to:

$$\varepsilon_p = \frac{|p_{\mathrm{CFD}} - p_{\mathrm{GNN}}|}{\max\left(p_{\mathrm{CFD}}\right)}. \tag{4.41}$$

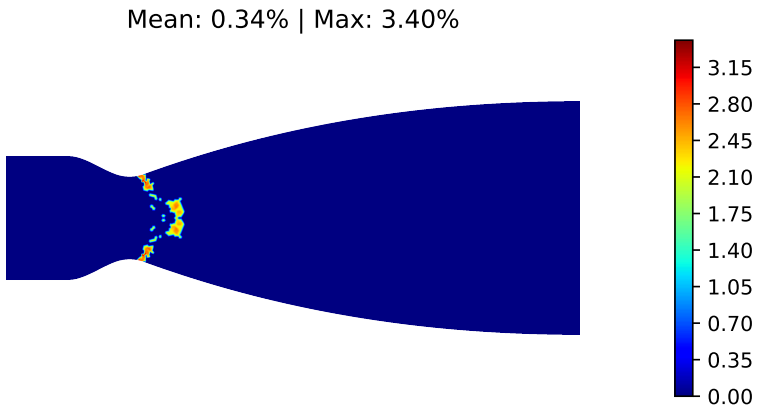
Overall, the consistency between the Mach and pressure results confirms the robustness of the proposed GNN architecture in modeling smooth nonlinear compressible flow configurations.



(a) CFD Mach target



(b) GNN Mach prediction



(c) Relative error map

Figure 4.13: Comparison of Mach number distribution for $\Pi = 0.20$

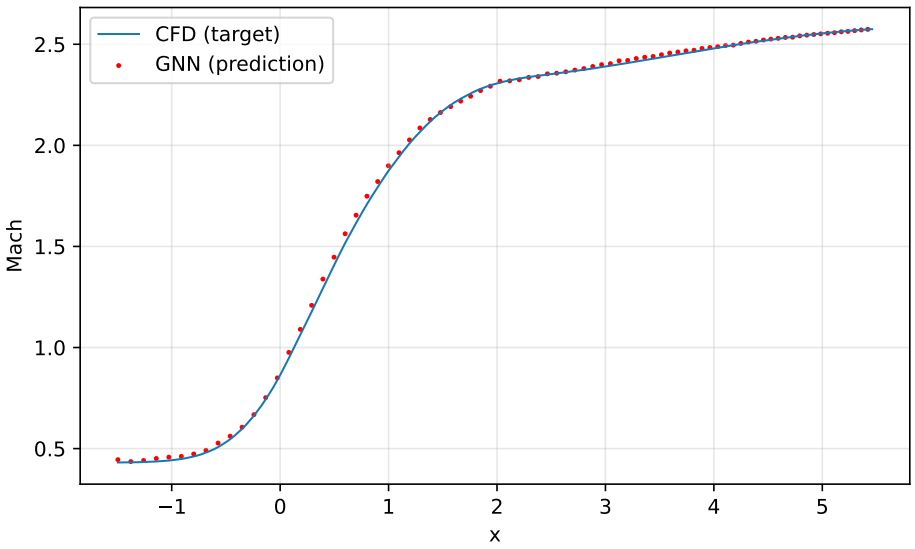
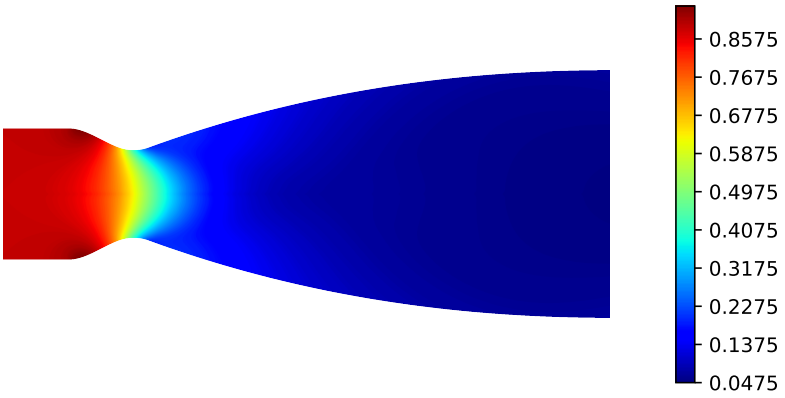
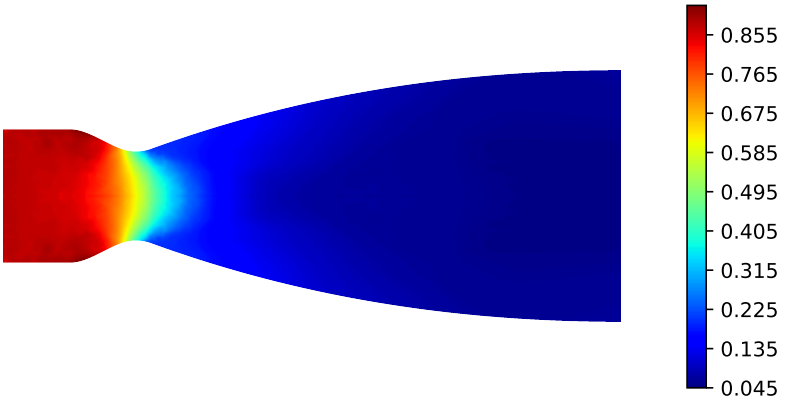


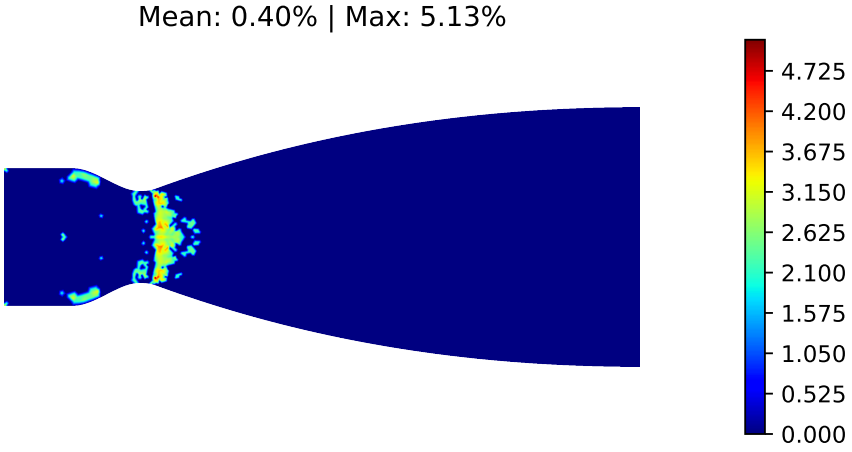
Figure 4.14: Centerline Mach profile



(a) CFD non-dimensional Pressure target



(b) GNN non-dimensional Pressure prediction



(c) Relative error map

Figure 4.15: Comparison of non-dimensional pressure distribution for $\Pi = 0.20$

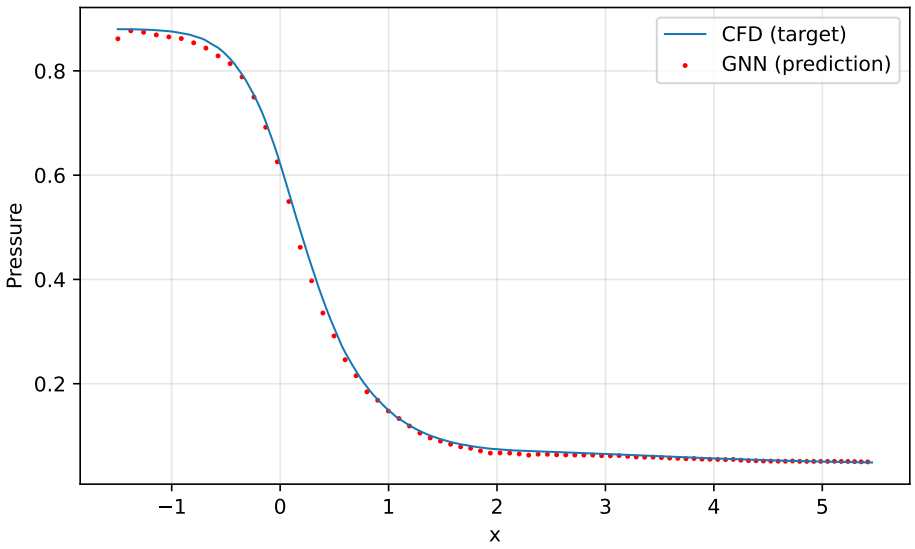


Figure 4.16: Centerline non-dimensional pressure profile

4.5 Discussion of Results

The analysis presented in this chapter shows that the proposed GNN-based surrogate model provides physically consistent and overall accurate predictions across all investigated flow regimes, including purely subsonic, subsonic–supersonic configurations with an internal normal shock, and fully supersonic cases. It is important to emphasize that the objective of the model is not to replace the CFD solver, but to generate an initial condition sufficiently close to the converged steady-state solution to accelerate and support the subsequent numerical simulation process. From this perspective, the obtained results can be considered satisfactory. In the purely subsonic regime, characterized by smooth solutions without discontinuities, the predicted Mach number and pressure fields reproduce the global structure of the reference CFD solution with good agreement. Nevertheless, the subsonic cases may exhibit non-negligible peak errors compared to fully supersonic configurations. This behavior, also reflected by the maximum-error trend reported as a function of the pressure ratio (Figure 4.17), can be attributed to the fact that subsonic solutions are more strongly influenced by the global pressure adjustment imposed by the boundary conditions and do not follow a monotonic spatial trend. As a result, the flow may display localized regions with relatively sharper gradients and a more heterogeneous spatial structure, which makes the regression task more demanding even in the absence of shocks. In the fully supersonic regime, the agreement between GNN predictions and CFD solutions is particularly strong. The flow becomes choked at the throat and accelerates monotonically downstream, leading to a smoother and more regular solution pattern. Moreover, once the flow is entirely supersonic inside the nozzle, the internal solution becomes insensitive to the imposed exit pressure: variations of the back pressure affect the downstream adjustment outside the nozzle, while the flow within the diverging section remains unchanged. Consistently, the maximum error remains limited in the fully supersonic configurations, confirming the robustness of the surrogate model when approximating smooth compressible flow fields. Greater challenges arise in the subsonic–supersonic regime with an internal normal shock. In this case, the presence of extremely steep gradients and near-discontinuous behavior leads to localized error amplification within the thin shock layer. The error maps and the maximum-error trend (Figure 4.17) confirm that the highest peaks are associated with shock-containing configurations, which is physically expected since continuous neural network approximators inherently struggle to represent sharp discontinuities. Importantly, however, the model preserves the global topology and physical coherence of the solution: it reproduces the upstream supersonic core, the downstream subsonic region,

and the overall jump across the shock. A remark is required regarding the interpretation of the maximum relative error in shock-containing cases, as it can vary significantly even within the same regime. In particular, the dispersion of peak errors can be attributed to the concurrent contribution of the following effects:

- **Mesh-induced shock thickness.** Depending on the local discretization, shocks forming closer to the throat appear more numerically smeared in the CFD reference, while shocks closer to the exit are sharper and confined to fewer cells. Since the GNN yields a continuous approximation, learning a distributed transition is intrinsically easier than reproducing a near-step discontinuity; therefore, sharper exit shocks tend to amplify localized mismatches and increase peak errors.

- **Increased downstream sensitivity.** As the shock approaches the nozzle exit, its position becomes more sensitive to the downstream pressure adjustment. Small variations in the predicted downstream state (or small discrepancies in the pressure field) can translate into a larger streamwise displacement of the shock location, increasing the likelihood of misalignment across a very thin shock layer.

- **Shock strength variation.** Shocks occurring closer to the throat develop from a lower upstream Mach level, producing a smaller jump in Mach number and pressure. Conversely, shocks located further downstream follow a longer supersonic acceleration and therefore exhibit a stronger discontinuity; for comparable positional errors, this yields larger pointwise mismatches and higher maximum relative errors.

For these reasons, peak errors should be interpreted together with domain-averaged metrics and spatial error maps, which consistently indicate that discrepancies remain strongly localized within steep-gradient regions. Overall, the results confirm that the proposed GNN architecture fulfills its intended purpose: generating flow fields that capture the essential physical structure of the solution and lie sufficiently close to the converged CFD state to serve as reliable initial conditions. Although localized discrepancies may arise in regions characterized by strong gradients or near-discontinuous behavior, these deviations remain limited in spatial extent and do not compromise the global fidelity of the predicted fields. The consistent behavior observed across fundamentally different flow regimes, together with the physically interpretable trends in the error distribution (Figure 4.17), supports the robustness and practical relevance of the proposed modeling approach within the range of operating conditions considered in this study.

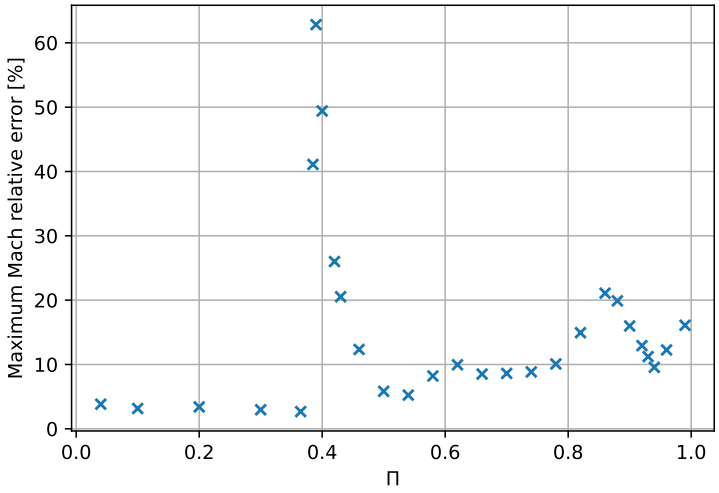
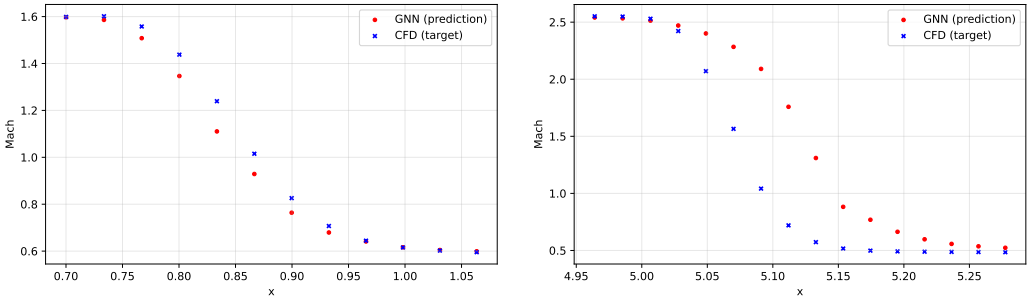


Figure 4.17: Maximum relative Mach error as a function of the pressure ratio Π



(a) $\Pi = 0.740$ (b) $\Pi = 0.385$

Figure 4.18: Zoomed centerline pressure profiles across the shock for two pressure-ratio configurations

Chapter 5

Parametric Prediction of Backward-Facing Step Flow

In the previous chapter, the proposed Graph Neural Network was employed as a predictive tool to accelerate the computation of steady compressible flows in a converging–diverging nozzle. The governing equations were the inviscid Euler equations, and the primary objective was not to fully replace the CFD solver, but rather to provide an informed initial condition capable of significantly reducing the number of iterations required for convergence. In the present chapter, we consider a fundamentally different physical configuration, namely the incompressible viscous flow over a backward-facing step. Unlike the previous case, viscous effects are explicitly taken into account through the Navier–Stokes equations, while compressibility effects are neglected. As a consequence, the flow remains continuous and no shock waves or discontinuities arise. The complexity of the problem is therefore not associated with mixed elliptic–hyperbolic behavior or shock formation, but rather with the nonlinear interaction between convection and diffusion, which leads to flow separation and recirculation phenomena downstream of the step. The parametric variability is introduced through the Reynolds number, which governs the relative importance of inertial and viscous forces. By varying the Reynolds number over a prescribed range, different flow regimes are obtained, characterized by changes in the size and structure of the recirculation zone and in the overall velocity field. This setup provides a suitable benchmark for assessing the ability of the Graph Neural Network to generalize across viscous flow regimes. A crucial difference with respect to the previous chapter concerns the role assigned to the neural model. Here, the objective is not merely to predict a flow field close to the steady-state solution in order to accelerate a subsequent CFD computation. Instead, the Graph Neural Network is evaluated in the perspective of acting as a surrogate solver. Con-

sequently, the emphasis is placed on minimizing the prediction error over the entire flow field, ensuring that the reconstructed velocity field accurately reproduce the numerical solution. To this end, a database of steady incompressible Navier–Stokes simulations for the backward-facing step configuration was generated over a range of Reynolds numbers. The trained model is then assessed in terms of accuracy, robustness, and generalization capability, with particular attention to its performance in predicting separation patterns and recirculation lengths as the Reynolds number varies.

5.1 Dataset Generation

5.1.1 Geometry and Mesh Generation

Geometry

The computational domain consists of a two-dimensional backward-facing step geometry adopted from the reference configuration presented in [1]. A schematic representation of the geometry is shown in Figure 5.1. The choice of this configuration is motivated by its widespread use as a benchmark test case for the validation of numerical methods in incompressible viscous flows.

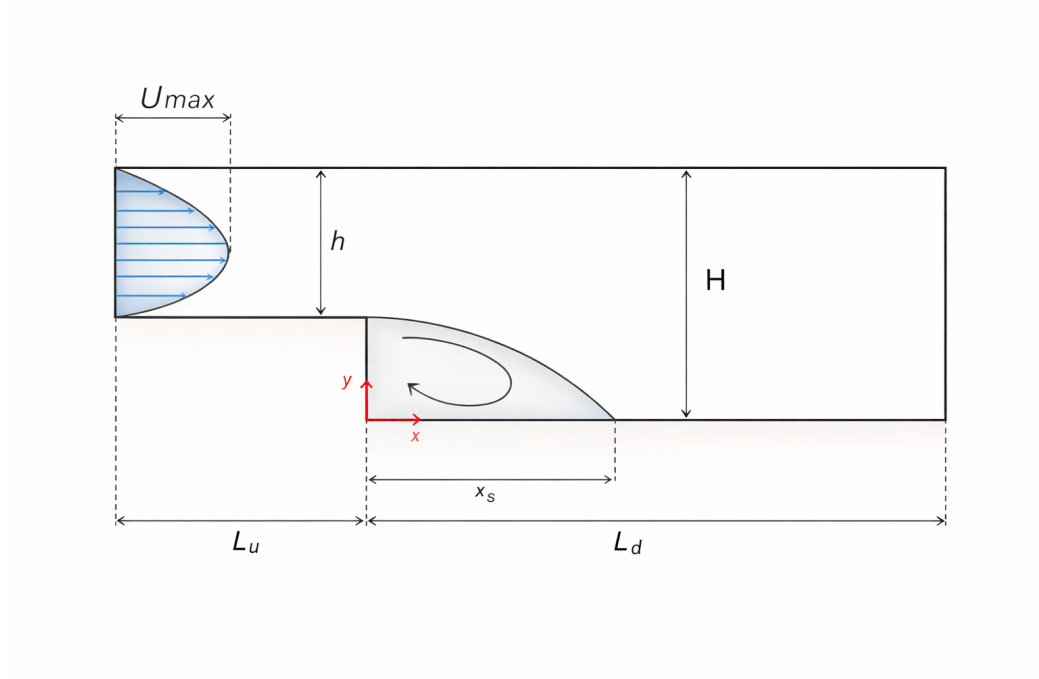


Figure 5.1: Backward-facing step geometry. Dimensions taken from [1]. Illustration by the author.

The geometry is fully defined by the following dimensional parameters:

$$h = 1\,\text{m} \quad \text{(step height)} \tag{5.1}$$
$$H = 1.9423\,\text{m} \quad \text{(downstream channel height)} \tag{5.2}$$
$$L_u = 5\,\text{m} \quad \text{(upstream length)} \tag{5.3}$$
$$L_d = 10\,\text{m} \quad \text{(downstream length)} \tag{5.4}$$

The upstream channel has height h, while downstream of the step the channel height increases to H. The expansion ratio is therefore defined as:

$$\frac{H}{h} = 1.9423 \tag{5.5}$$

The upstream length L_u ensures the development of a fully established velocity profile before the step, while the downstream length L_d is chosen sufficiently large to allow the flow to reattach and the recirculation region to fully develop. Although the geometry can be represented in three dimensions, the present study adopts a two-dimensional formulation. This assumption is consistent with the reference analysis reported in [1], where it is shown that, within the Reynolds number range considered in this work, no significant three-dimensional instabilities arise that would alter the mean flow solution compared to experimental observations. In particular, for moderate Reynolds numbers, the backward-facing step flow remains predominantly two-dimensional in its averaged structure, with recirculation length and velocity profiles accurately captured by two-dimensional simulations. Therefore, under the present operating conditions, the two-dimensional approximation does not introduce a loss of physical fidelity while significantly reducing computational cost and simplifying dataset generation.

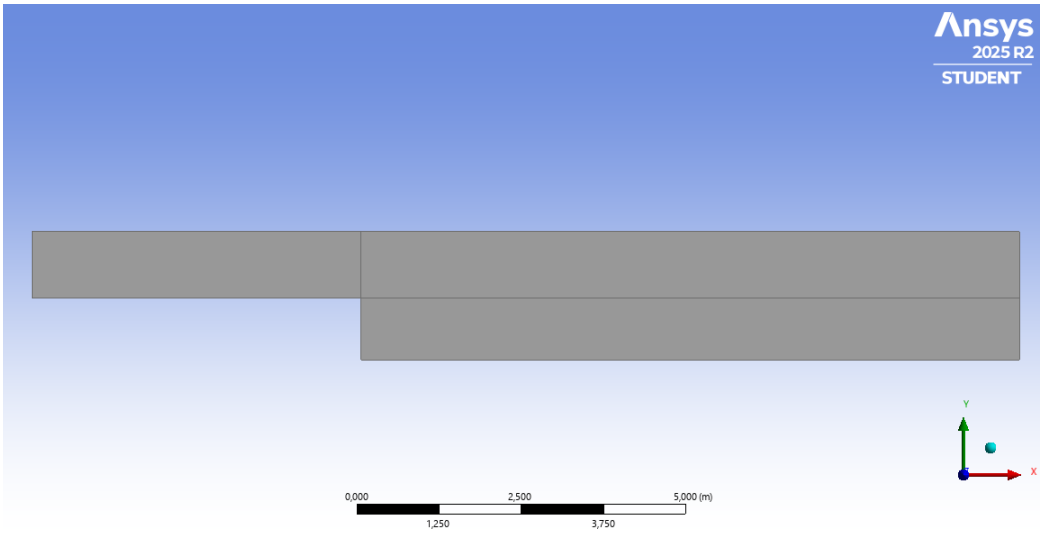


Figure 5.2: Two-dimensional computational domain of the backward-facing step geometry

Mesh Generation

The computational domain is discretized using a structured quadrilateral mesh, as illustrated in Figure 5.3. The grid is aligned with the streamwise and wall-normal directions in order to accurately capture boundary-layer development and recirculation phenomena typical of backward-facing step flows. A locally refined region is introduced in the vicinity of the step and along the downstream wall, where strong velocity gradients and flow separation occur. In particular, the mesh is progressively refined near the reattachment zone to ensure an accurate prediction of the recirculation length and wall shear stress distribution. Away from the step, a gradually coarsened mesh is employed in order to reduce computational cost while maintaining sufficient resolution in regions of lower gradient intensity. Unlike the compressible nozzle case presented in the previous chapter, no shock-capturing considerations are required here, since the incompressible Navier–Stokes formulation leads to a continuous solution field. The mesh design is therefore driven primarily by the need to resolve boundary layers and separation regions rather than discontinuities. The upstream region is discretized with a uniform resolution to properly resolve the developing velocity profile before the step, while the downstream length is sufficiently resolved to capture the entire recirculation bubble and the subsequent flow recovery. This strategy ensures a balanced compromise between numerical accuracy and computational efficiency across the range of Reynolds numbers considered in this study.

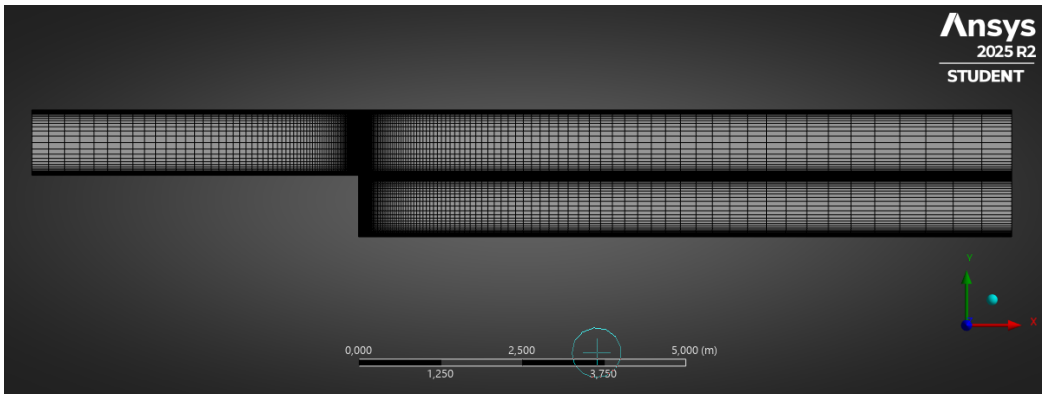


Figure 5.3: Structured computational mesh of the backward-facing step domain

5.1.2 Numerical Discretisation

The flow is modeled by the two-dimensional incompressible Navier–Stokes equations for a Newtonian viscous fluid, neglecting body-force terms. In conservative form, the governing equations read:

$$\begin{cases} \nabla \cdot \mathbf{u} = 0 \\ \dfrac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\dfrac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} \end{cases} \tag{5.6}$$

where $\mathbf{u} = (u, v)$ denotes the velocity field, p the pressure, ρ the fluid density, and ν the kinematic viscosity. Unlike the compressible formulation considered in the previous chapter, the present model explicitly accounts for viscous effects while neglecting compressibility. As a consequence, the flow field remains continuous and no shock waves or discontinuities arise within the computational domain. The numerical simulations were performed using *ANSYS Fluent*. A pressure-based solver was adopted, consistent with the incompressible formulation of the governing equations. The pressure–velocity coupling was handled within the standard segregated framework of the solver. The flow was modeled as laminar, without introducing any turbulence model. This modeling choice is supported by the experimental study taken as reference [1], which reports that for Reynolds numbers below $Re = 300$ the backward-facing step flow remains laminar and does not exhibit three dimensional instabilities or transition to turbulence. Therefore, within the Reynolds number range considered in this work, a laminar formulation is physically consistent and sufficient to reproduce the relevant flow dynamics. At the inlet boundary, a fully developed parabolic velocity profile was prescribed, with a mean velocity equal to $U_{\text{mean}} = 1\,\text{m/s}$. The imposed profile is:

$$u(y) = U_{\max}\left(\frac{y - Y_{\min}}{H} - \left(\frac{y - Y_{\min}}{H}\right)^2\right) \tag{5.7}$$

where $H = Y_{\max} - Y_{\min}$ denotes the channel height. The maximum velocity is defined as

$$U_{\max} = 6\,U_{\text{mean}} \tag{5.8}$$

ensuring that the spatial average of the inlet profile equals 1 m/s. The profile was implemented in *ANSYS Fluent* through a User-Defined Function (UDF), allowing precise control over the inlet boundary condition. The choice of prescribing an already developed parabolic velocity profile at the inlet was motivated by computational considerations. If a uniform velocity profile had been

imposed, a significantly longer upstream section would have been required to allow the flow to naturally develop before reaching the step. By directly prescribing a fully developed laminar profile, the upstream length can be reduced while maintaining physical consistency, since the flow approaching the backward-facing step is representative of a developed channel flow. This strategy enables a more compact computational domain without affecting the accuracy of the separation and recirculation phenomena downstream of the step. Eventually the steady-state solution was obtained through the iterative convergence of the pressure-based solver. Since the incompressible viscous formulation does not involve acoustic wave propagation, the numerical stability is not restricted by the compressible CFL condition described in the previous chapter. However, convergence still depends on the nonlinear coupling between pressure and velocity and on the resolution of the recirculation region downstream of the step, which becomes increasingly sensitive as the Reynolds number increases.

5.1.3 Parametric CFD Simulations and Data Pre – Procssing

A total of 31 steady-state simulations were performed. In each simulation, the inlet mean velocity was kept constant at $U_{\text{mean}} = 1\,\text{m/s}$, while the Reynolds number was progressively varied in the range $1 \leq Re \leq 300$. The Reynolds number is defined as

$$Re = \frac{\rho\, U_{\text{mean}}\, h}{\mu} \tag{5.9}$$

where ρ denotes the fluid density, U_{mean} the mean inlet velocity, h the step height, and μ the dynamic viscosity. In the present study, the Reynolds number was varied by modifying the fluid density while keeping both the viscosity and inlet velocity fixed. This approach allows isolating the influence of inertial effects relative to viscous forces without altering the imposed boundary conditions. The resulting dataset therefore captures the progressive transition from strongly diffusion-dominated regimes at low Reynolds numbers to increasingly inertia-dominated laminar regimes as Re approaches 300. In particular, variations in the size of the recirculation bubble and in the downstream reattachment length are observed across the parametric sweep.
The numerical simulations were carried out in *ANSYS Fluent*, and the solution fields were exported in a vertex-centered format. The primary physical quantities at each node include the velocity components and pressure, defined directly at the mesh vertices. The exported text files were subsequently processed using a dedicated preprocessing script, which reconstructed the graph

structure associated with each simulation. This procedure transforms the raw CFD outputs into structured graph datasets suitable for message-passing neural network training.

5.2 Graph-based representation and Message – Passing architecture

5.2.1 Graph construction

For each simulation, a graph $G = (V, E)$ was constructed, where:

- V represents the set of mesh vertices,
- E represents the set of edges defined according to mesh connectivity.

Each simulation corresponds to one graph sample within the dataset. Each node $v_i \in V$ is associated with the following feature vector:

$$\mathbf{x}_i = [x,\ y,\ \text{inlet},\ \text{outlet},\ \text{wall},\ \text{fluid}] \tag{5.10}$$

where:

- x, y are the spatial coordinates,
- inlet, outlet, wall, fluid are binary indicators encoding the boundary condition type

This representation embeds both geometric information and boundary condition encoding directly into the nodal feature space. In addition to the nodal attributes, geometric relationships between neighboring nodes were explicitly encoded at the edge level. Given an edge connecting node i (source) and node j (destination), the corresponding edge features were constructed using relative geometric quantities derived from the nodal descriptors. The relative displacement vector was defined as

$$\Delta\mathbf{x}_{ij} = \begin{bmatrix} x_j - x_i \\ y_j - y_i \end{bmatrix} \tag{5.11}$$

with associated Euclidean distance:

$$d_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \tag{5.12}$$

The complete edge feature vector was therefore defined as:

$$\mathbf{e}_{ij} = [\Delta x_{ij}, \Delta y_{ij}, d_{ij}] \tag{5.13}$$

This formulation allows the Graph Neural Network to incorporate relative spatial and geometric information during message passing, consistently with the local interaction structure of the underlying conservation laws.

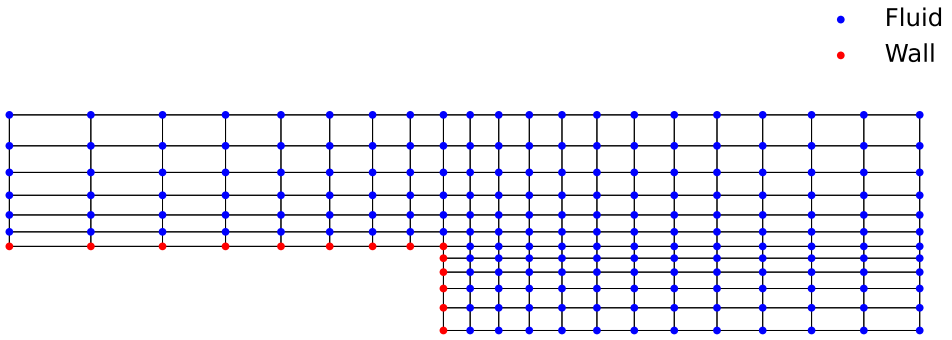


Figure 5.4: Graph representation of a zoomed-in region of the backward-facing step computational domain

5.2.2 Design of the Message Passing Architecture

The message passing architecture adopted for the backward-facing step configuration closely follows the encoder–message passing–decoder structure introduced in Chapter 4. For this reason, the full formulation is not repeated here, and only the key differences with respect to the compressible nozzle case are highlighted. In the present incompressible viscous setting, the dimensionality of the initial graph features differs from the previous configuration. Node features are defined as $\mathbf{x}_i \in \mathbb{R}^6$, while edge features belong to $\mathbf{e}_{ij} \in \mathbb{R}^3$. As in the previous chapter, both node and edge features are independently projected into a latent space of dimension $H = 128$ through learnable multilayer perceptrons:

$$\mathbf{h}_i^{(0)} = \phi_{\text{enc}}(\mathbf{x}_i), \qquad \tilde{\mathbf{e}}_{ij} = \psi_{\text{enc}}(\mathbf{e}_{ij}) \tag{5.14}$$

The overall latent dimensionality and the depth of the network are kept unchanged to ensure consistency in model capacity and facilitate comparison between the two physical configurations. A second fundamental difference concerns the global conditioning mechanism. In Chapter 4, the latent node embeddings were modulated using the pressure ratio $\mathbf{\Pi}$. In the present case, the conditioning variable is the Reynolds number $\mathbf{Re}$, which parametrizes the viscous flow regime and directly governs the strength of inertial versus viscous effects. Two learnable transformations are introduced:

$$\boldsymbol{\gamma} = \tanh(W_\gamma \,\mathrm{Re}), \qquad \boldsymbol{\beta} = W_\beta \,\mathrm{Re} \tag{5.15}$$

and the encoded node features are updated as

$$\mathbf{h}_i^{(0)} = \boldsymbol{\gamma} \odot \mathbf{h}_i^{(0)} + \boldsymbol{\beta}. \tag{5.16}$$

This modulation enables the network to adapt its internal representation to different viscous regimes while preserving a shared geometric structure of the computational domain. The message computation, mean aggregation operator, residual update rule and decoding stage remain unchanged with respect to the previous chapter. Consequently, The resulting model can be interpreted as a Reynolds-conditioned residual Message Passing Neural Network operating on graph representations in which the only geometric information is provided by node coordinates.

5.3 Training Procedure

The training strategy adopted in Chapter 4 was retained in the present analysis in terms of dataset splitting, optimization algorithm, learning-rate scheduling, gradient clipping, and model selection criteria. The graph dataset was divided into training and validation subsets using the same 80%–20% split, and optimization was performed using the Adam optimizer with identical hyperparameters.

In order to assess the impact of the loss definition on model accuracy, two independent training configurations were considered for the same GNN architecture. In the first configuration, the network was trained using a standard node-wise mean-squared error (MSE) computed in normalized space. In the second configuration, a relative-error contribution was added to the baseline MSE. The motivation for introducing the relative term and its effect on the predictions will become clear in the following Results section.

Let N denote the number of nodes and C the number of predicted channels. Let $\mathbf{y}_i \in \mathbb{R}^C$ and $\hat{\mathbf{y}}_i \in \mathbb{R}^C$ be the reference and predicted output vectors at node i, respectively. In normalized space, these are defined as

$$\tilde{\mathbf{y}}_i = \frac{\mathbf{y}_i - \boldsymbol{\mu}_y}{\boldsymbol{\sigma}_y}, \qquad \tilde{\hat{\mathbf{y}}}_i = \frac{\hat{\mathbf{y}}_i - \boldsymbol{\mu}_y}{\boldsymbol{\sigma}_y}, \tag{5.17}$$

where $\boldsymbol{\mu}_y \in \mathbb{R}^C$ and $\boldsymbol{\sigma}_y \in \mathbb{R}^C$ are the dataset mean and standard deviation (applied channel-wise). The baseline data loss adopted in both configurations is the normalized node-wise MSE:

$$\mathcal{L}_{\text{data}} = \frac{1}{NC} \sum_{i=1}^{N} \sum_{c=1}^{C} \left(\tilde{\hat{y}}_{i,c} - \tilde{y}_{i,c} \right)^2. \tag{5.18}$$

In the second configuration, the MSE term was augmented with a relative-error loss computed over all channels:

$$\mathcal{L}_{\text{rel}} = \frac{1}{NC} \sum_{i=1}^{N} \sum_{c=1}^{C} \left(\frac{\hat{y}_{i,c} - y_{i,c}}{|y_{i,c}| + \varepsilon} \right)^2, \tag{5.19}$$

where ε is a small positive constant introduced to avoid numerical issues when $y_{i,c}$ approaches zero. The total loss for the second configuration is therefore

$$\mathcal{L}_{\text{tot}} = \mathcal{L}_{\text{data}} + \mathcal{L}_{\text{rel}}, \tag{5.20}$$

where $\mathcal{L}_{\text{data}}$ is the baseline node-wise MSE and $\mathcal{L}_{\text{rel}}$ is a weighted relative-error term. In the present implementation, the relative loss is computed with

a wall-distance weight $w(d)$ that assigns larger importance to nodes close to the bottom wall and smoothly reduces the weight toward the bulk region, thereby prioritizing near-wall accuracy. As in the previous training configuration, optimization was carried out in normalized space, while monitoring and model selection were performed using denormalized physical-scale metrics.

5.4 Results

This section presents the evaluation of the proposed graph-based model for the backward-facing step flow across different Reynolds-number conditions. The analysis aims to assess both the qualitative and quantitative agreement between the GNN predictions and the reference CFD solutions. To provide a physically meaningful interpretation of the model performance, the results are organized according to three representative operating conditions, corresponding to a low, intermediate, and high Reynolds number. These regimes exhibit distinct flow features, including variations in the size and intensity of the recirculation bubble and in the downstream reattachment behavior. To quantify the effect of the training objective, the results are reported for two independently trained models employing the same architecture and training settings but different loss functions: a baseline MSE formulation and an augmented MSE + relative-error formulation. All graph samples presented in this section belong to the validation subset and were not used during training; therefore, the comparisons shown below evaluate the generalization capability of the network on previously unseen flow configurations. For each Reynolds-number condition and for each loss configuration, a comparison between the target CFD field and the predicted field is presented, together with the corresponding error distribution. Additional quantitative metrics are used to evaluate the global predictive performance across the entire validation set.

Results with MSE loss

This section discusses the predictive performance of the baseline model trained using the standard normalized MSE loss. The analysis is first conducted on primary flow quantities through full-field comparisons and global error metrics. Subsequently, a more stringent assessment is carried out by inspecting near-wall-sensitive quantities, highlighting the limitations of the MSE-only formulation in accurately capturing velocity gradients at the wall. Figures 5.5 – 5.6 – 5.7 compare the predicted (GNN) and reference (CFD) distributions of the streamwise velocity component u for three representative validation

cases ($Re = 10$, $Re = 160$ and $Re = 300$). From a qualitative standpoint, the agreement is excellent across the entire domain: the overall acceleration in the upper channel, the low-momentum region downstream of the step, and the progressive downstream recovery are all reproduced with nearly indistinguishable contour patterns between prediction and target. For each case, the prediction error was quantified through a normalized absolute error, computed as the pointwise difference between reference and prediction divided by the maximum reference velocity in the corresponding CFD sample. Denoting by u_{CFD} the CFD reference and by u_{GNN} the network prediction, the local normalized error is defined as

$$\varepsilon_u = \frac{|u_{\mathrm{CFD}} - u_{\mathrm{GNN}}|}{\max(u_{\mathrm{CFD}})}. \tag{5.21}$$

The mean and maximum percentage errors are computed as

$$\bar{\varepsilon}_{\%} = 100 \cdot \frac{1}{N} \cdot \sum_{i=1}^{N} \varepsilon_i, \qquad \varepsilon_{\%,\max} = 100 \cdot \max_{i=1,\dots,N}(\varepsilon_i) \tag{5.22}$$

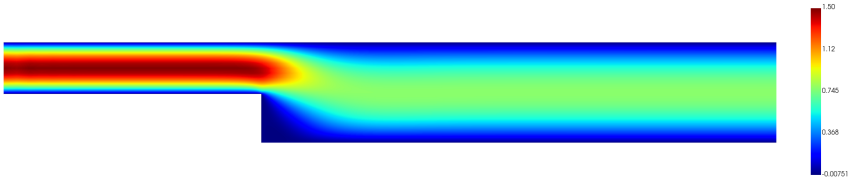
with N the number of nodes. The resulting values for the three validation cases are summarized in Table 5.1.

Table 5.1: Validation errors for u with mean and maximum normalized absolute error (in %).

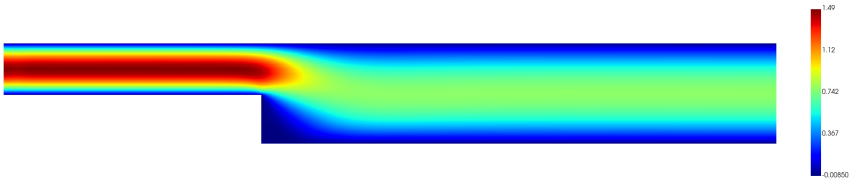
Re	$\bar{\varepsilon}_{\%}$ [%]	$\varepsilon_{\%,\max}$ [%]
10	**0.070**	**0.830**
160	**0.037**	**0.261**
300	**0.068**	**0.450**

Despite the apparently near-perfect match in the full-field contours, a more stringent assessment reveals that the baseline MSE formulation does not guarantee an accurate reconstruction of near-wall gradients. This limitation becomes evident when inspecting the wall-normal derivative $\partial u/\partial y$ evaluated along the bottom wall, reported in Fig. 5.8 for the same three Reynolds numbers. In all cases, the GNN estimates exhibit a significantly larger scatter and reduced consistency with respect to the CFD trend, particularly in the region downstream of the step where separation and reattachment occur and where the near-wall shear varies sharply. To further highlight this issue, Figures 5.9 – 5.10 – 5.11 reports a comparison of the velocity profile extracted at a fixed streamwise location ($x = 0.2$) for the representative low-Reynolds case ($Re = 10$). When displayed over the full cross-section (Figure 5.9), the

prediction follows the reference profile closely, suggesting an excellent agreement. However, a zoom-in on the near-wall region (Figure 5.11) reveals a non-negligible discrepancy within the boundary layer, which cannot be considered acceptable when wall-related quantities are of interest. Since the wall shear stress scales with the wall velocity gradient, $\tau_w = \mu \left.\dfrac{\partial u}{\partial y}\right|_w$, even small absolute deviations in u within the first near-wall nodes may translate into large errors in $\partial u/\partial y$ and in derived quantities such as skin friction and friction-related metrics. These observations motivate the introduction of a relative-error contribution in the loss function (discussed in the next section), aimed at improving robustness across different magnitudes of the solution and, most importantly, enhancing the predictive accuracy in physically critical near-wall regions. For completeness, additional pressure-field comparisons are reported in Appendix A, in order to avoid overloading the present section with redundant contour plots.

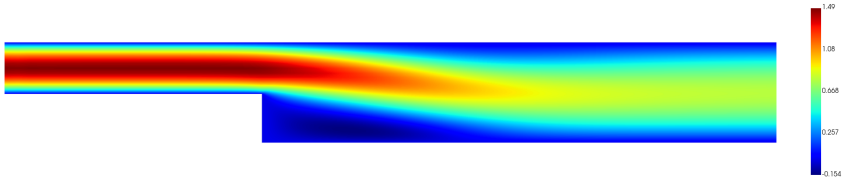


(a) CFD target

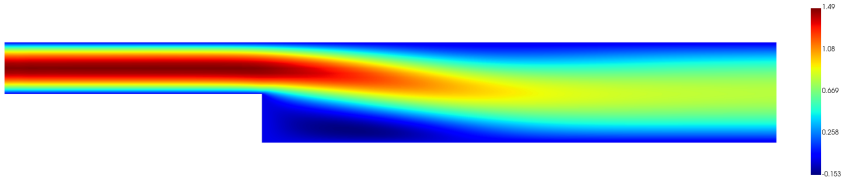


(b) GNN prediction

Figure 5.5: Streamwise velocity field u for the validation case at $Re = 10$. CFD reference (top) vs GNN prediction (bottom)

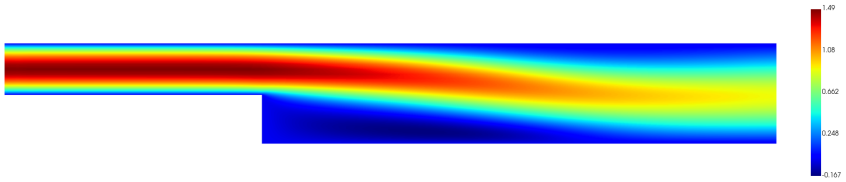


(a) CFD target

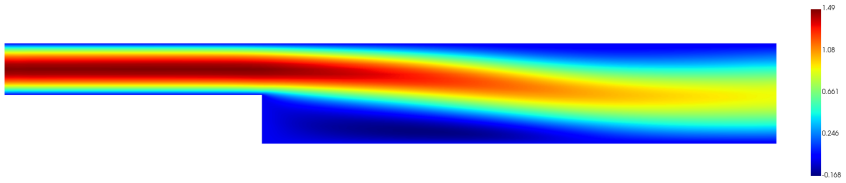


(b) GNN prediction

Figure 5.6: Streamwise velocity field u for the validation case at $Re = 160$. CFD reference (top) vs GNN prediction (bottom)

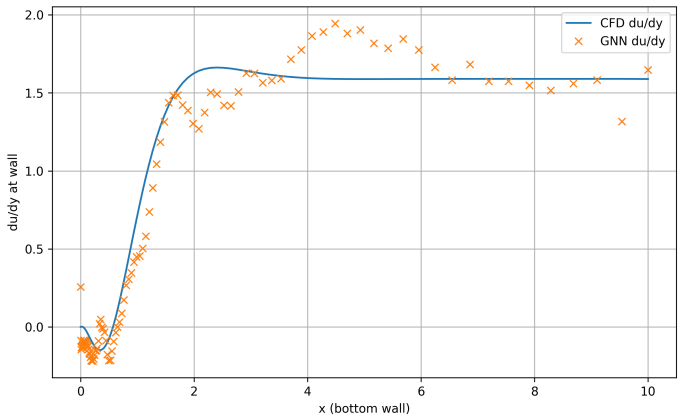


(a) CFD target

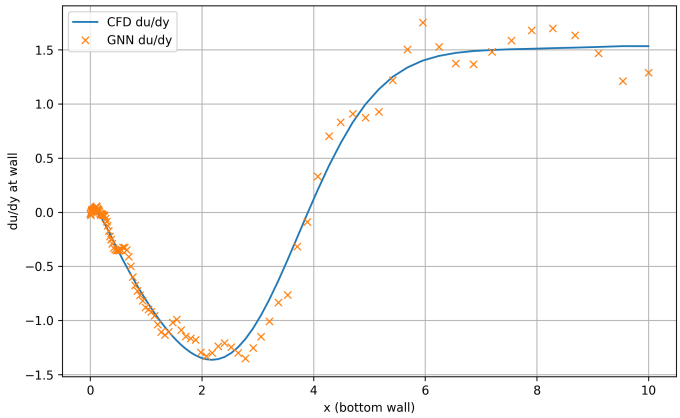


(b) GNN prediction

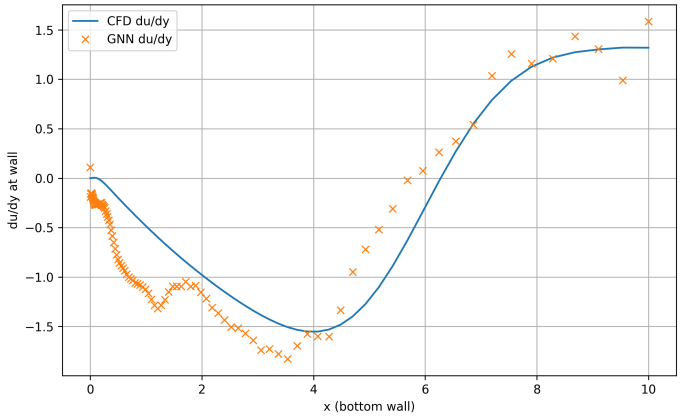
Figure 5.7: Streamwise velocity field u for the validation case at $Re = 300$. CFD reference (top) vs GNN prediction (bottom)



(a) $Re = 10$



(b) $Re = 160$



(c) $Re = 300$

Figure 5.8: Comparison of $\partial u/\partial y$ evaluated along the bottom wall between CFD reference and GNN prediction for three validation cases.

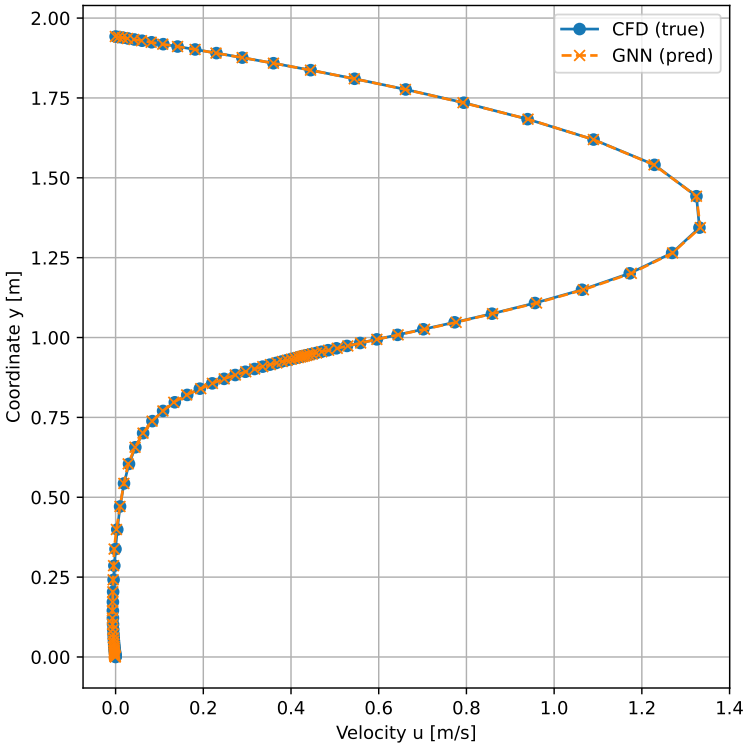


Figure 5.9: Streamwise velocity profile at $x = 0.2$ for the $Re = 10$ validation case, CFD vs GNN.

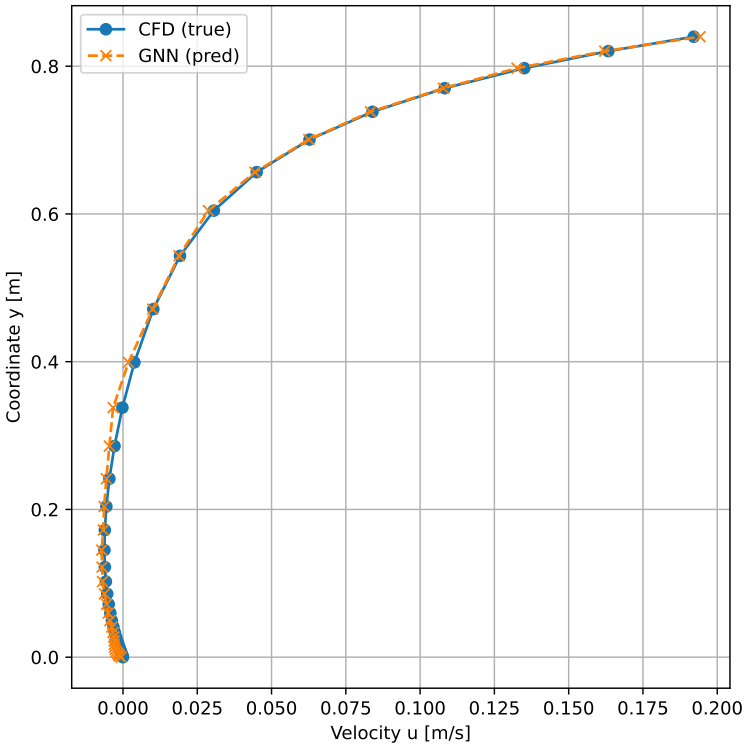


Figure 5.10: Streamwise velocity profile at $x = 0.2$ ($Re = 10$) shown with an intermediate zoom, CFD vs GNN.

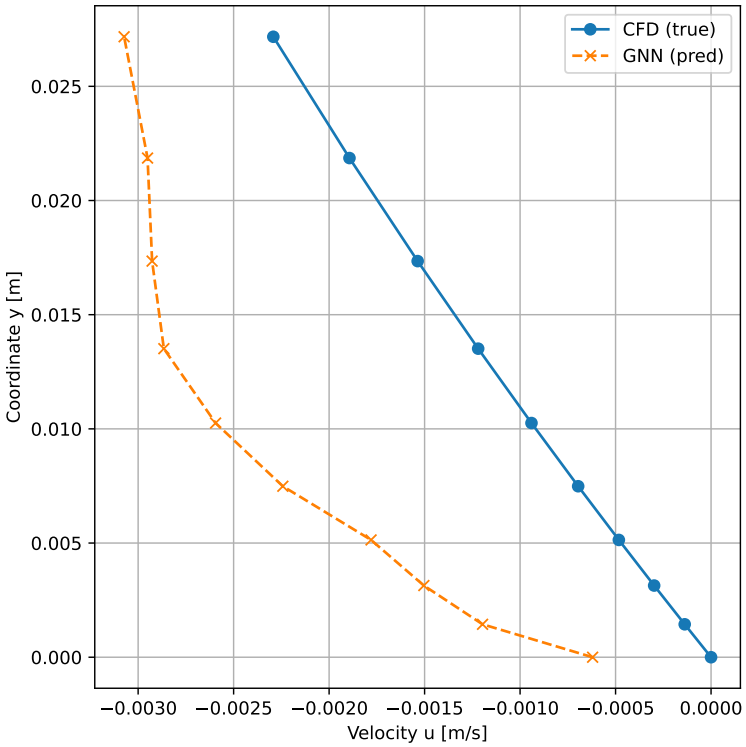


Figure 5.11: Near-wall zoom of the streamwise velocity profile at $x = 0.2$ ($Re = 10$) highlighting boundary-layer discrepancies affecting $(\partial u/\partial y)_w$.

Augmented training with relative-error loss

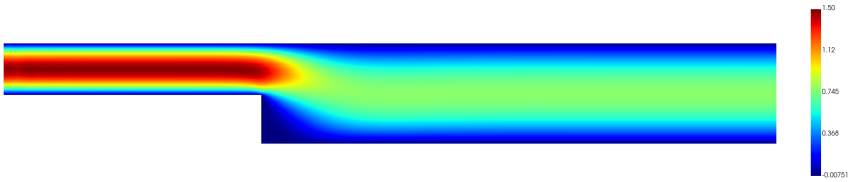
The baseline MSE-only training provides an accurate reconstruction of the bulk flow; however, it does not explicitly distinguish between errors occurring in regions characterized by very different magnitudes. This limitation is particularly relevant for boundary-layer dynamics. As shown by the velocity profiles discussed in the previous section, the characteristic order of magnitude of u in the core region is $\mathcal{O}(1)$, whereas within the boundary layer it can decrease to $\mathcal{O}(10^{-3})$. Consequently, the same absolute deviation (e.g. 10^{-3}) corresponds to a negligible relative error in the bulk (about 0.1%), while it represents about an 100% relative discrepancy close to the wall. Since a pure MSE formulation weights both situations equally in terms of absolute error, the optimizer has no incentive to prioritize accuracy in low-magnitude but physically critical near-wall regions. For this reason, a relative-error contribution was added to the loss function in order to better balance the training objective across the domain. Figures 5.12 – 5.13 – 5.14 report the comparison of the u field for three representative validation cases ($Re = 10$, $Re = 160$, and $Re = 300$). The global structure of the predicted fields remains essen-

tially unchanged with respect to the MSE-only model, confirming that the augmented objective does not deteriorate the bulk-flow reconstruction. The resulting values of the normalized absolute error for the three validation cases are summarized in Table 5.2.

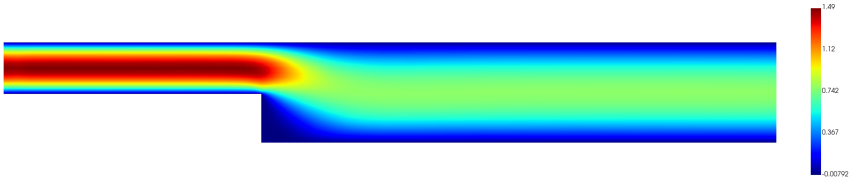
Table 5.2: Validation errors for u with mean and maximum normalized absolute error (in %).

Re	$\bar{\varepsilon}_{\%}$ [%]	$\varepsilon_{\%,\max}$ [%]
10	**0.075**	**0.800**
160	**0.032**	**0.327**
300	**0.061**	**0.551**

The main improvement introduced by the relative-error term becomes evident when inspecting near-wall-sensitive quantities. Figure 5.15 shows the wall-normal gradient $\partial u_x/\partial y$ evaluated along the bottom wall. Compared to the baseline configuration, the agreement with the CFD reference is markedly improved for all Reynolds numbers, with a substantial reduction of the scatter and a much better reconstruction of both the amplitude and the streamwise evolution of the gradient in the separated-flow region. This behavior is further confirmed by the velocity profile extracted at $x = 0.2$ for the representative case $Re = 10$ (Figures 5.16 – 5.17 – 5.18). While the full cross-section comparison remains excellent, the near-wall zoom reveals that the boundary layer is now resolved with significantly higher fidelity. This improvement is crucial because wall-related quantities, such as the wall shear stress, are directly determined by the near-wall velocity gradient. Overall, the augmented loss formulation allows the model to retain the high accuracy observed in the bulk while substantially improving the prediction of boundary-layer dynamics and wall gradients.

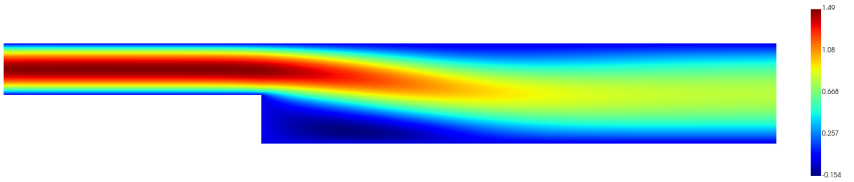


(a) CFD target

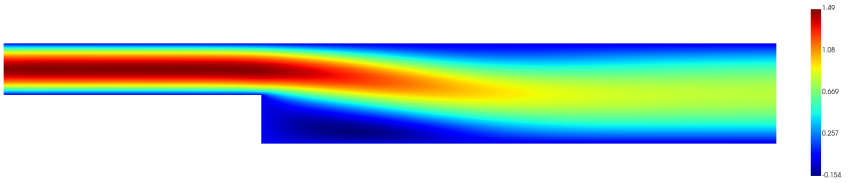


(b) GNN prediction

Figure 5.12: Streamwise velocity field u for the validation case at $Re = 10$. CFD reference (top) vs GNN prediction (bottom)

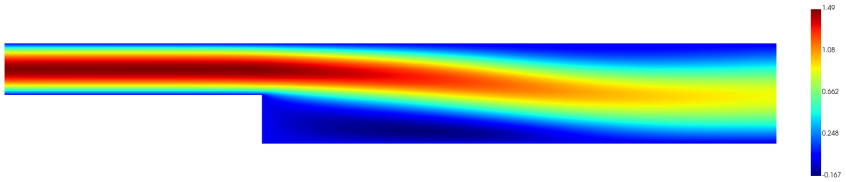


(a) CFD target

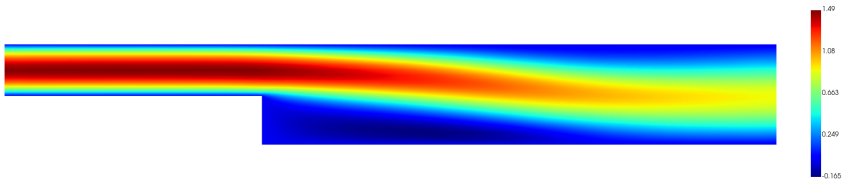


(b) GNN prediction

Figure 5.13: Streamwise velocity field u for the validation case at $Re = 160$. CFD reference (top) vs GNN prediction (bottom)

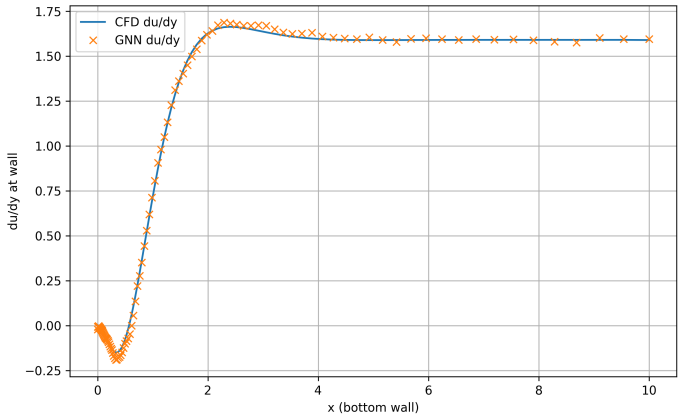


(a) CFD target

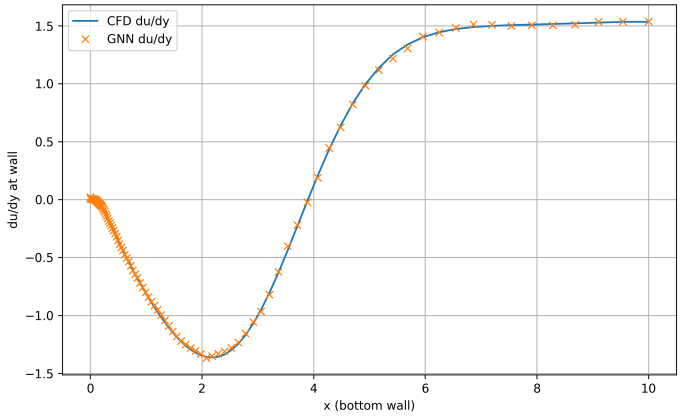


(b) GNN prediction

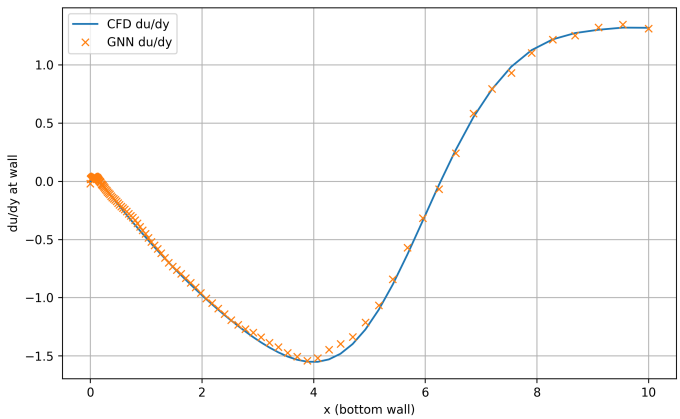
Figure 5.14: Streamwise velocity field u for the validation case at $Re = 300$. CFD reference (top) vs GNN prediction (bottom)



(a) $Re = 10$



(b) $Re = 160$



(c) $Re = 300$

Figure 5.15: Comparison of $\partial u/\partial y$ evaluated along the bottom wall between CFD reference and GNN prediction for three validation cases.

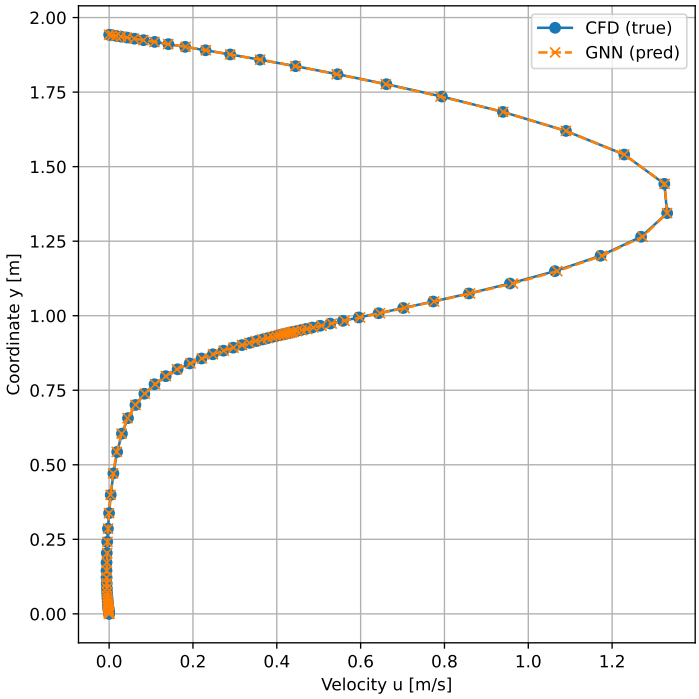


Figure 5.16: Streamwise velocity profile at $x = 0.2$ for the $Re = 10$ validation case, CFD vs GNN.

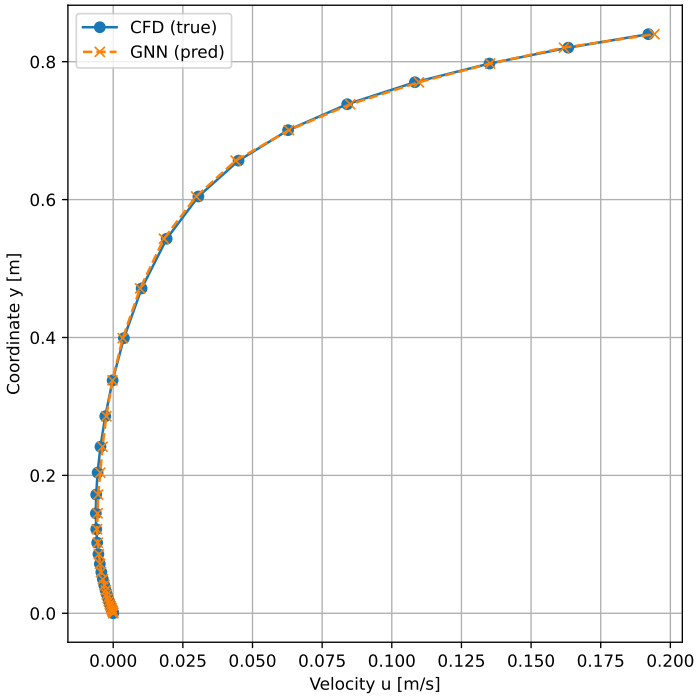


Figure 5.17: Streamwise velocity profile at $x = 0.2$ ($Re = 10$) shown with an intermediate zoom, CFD vs GNN.

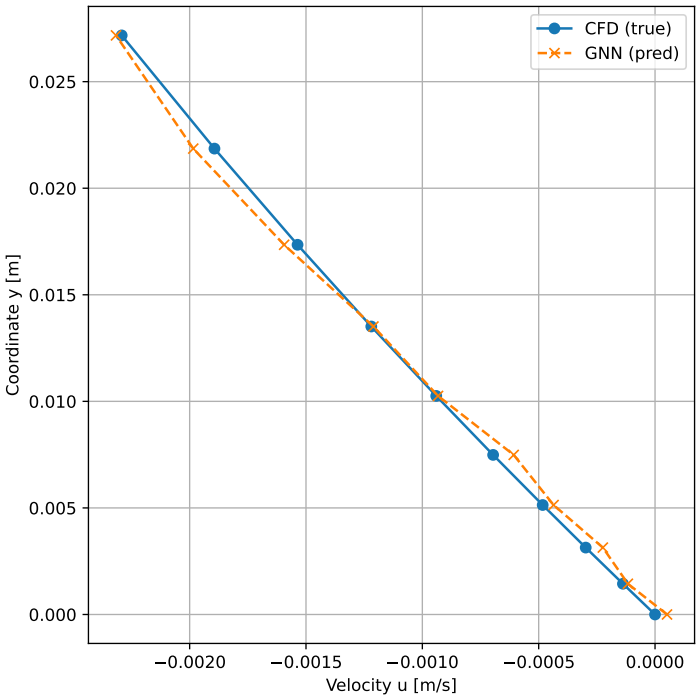


Figure 5.18: Near-wall zoom of the streamwise velocity profile at $x = 0.2$ ($Re = 10$) highlighting boundary-layer discrepancies affecting $(\partial u/\partial y)_w$.

5.5 Discussion of Results

This chapter investigated the use of a graph-based surrogate model for the backward-facing step configuration across different Reynolds numbers. The results showed that the proposed message-passing architecture can accurately reproduce the global structure of the flow fields, particularly for the streamwise velocity component and the pressure distribution. However, the analysis also highlighted that a purely MSE-based training objective, while effective on bulk quantities, may underemphasize low-magnitude yet physically critical near-wall regions. This limitation becomes evident when inspecting wall-sensitive quantities such as the wall-normal velocity gradient and, consequently, wall shear stress. Introducing a relative-error contribution in the loss function preserved the excellent agreement on the global fields while substantially improving the reconstruction of boundary-layer dynamics and wall-gradient trends. In addition to the physical assessment, a dataset-size sensitivity study was performed to evaluate how the amount of available data affects generalization. In this experiment, the model was trained using progressively larger datasets, each subsequently split into training and validation subsets with the same 80%–20% ratio adopted throughout this work. After training, a mean MSE was computed by evaluating the prediction error over the full collection of 31 graphs.

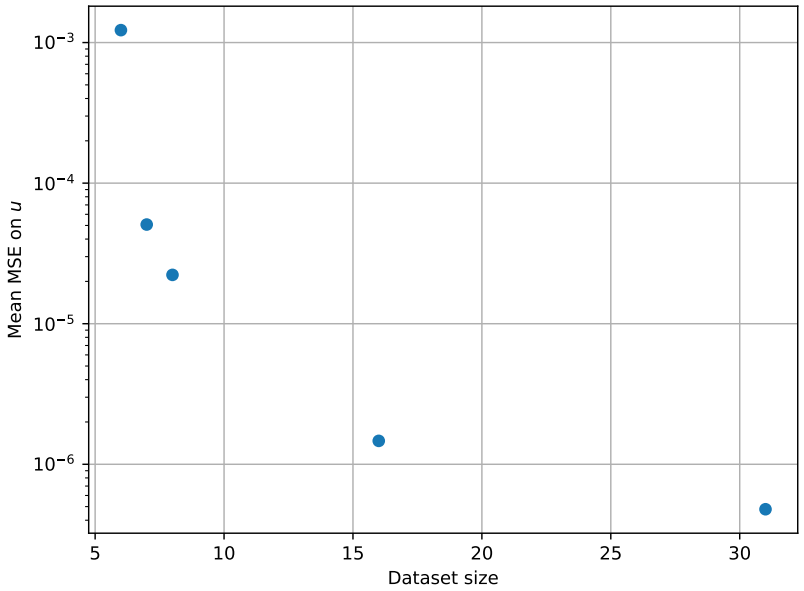


Figure 5.19: Mean MSE evaluated on the full set of 31 graphs as a function of the total number of available graphs used to form the dataset

Figure 5.19 shows that increasing the dataset size leads to a systematic reduction of the average MSE, while training with a limited number of available graphs results in significantly larger errors when assessed on the complete set. This trend confirms that surrogate accuracy strongly depends on dataset coverage and diversity, and it motivates the use of larger datasets (or data augmentation strategies) when robust generalization across operating conditions is required.

Bibliography

[1] Gautam Biswas, Michael Breuer, and Franz Durst. Backward-facing step flows for various expansion ratios at low and moderate reynolds numbers. *J. Fluids Eng.*, 126(3):362–374, 2004.

Chapter 6

Conclusions

This thesis investigated the use of Graph Neural Networks (GNNs) as surrogate models for computational fluid dynamics problems governed by fundamentally different physical assumptions and mathematical properties. The main objective was to assess whether the same message-passing architecture, trained independently on problem-specific datasets, can successfully approximate flow solutions arising from distinct governing equations, while retaining a level of physical fidelity that is meaningful for engineering analysis. Two canonical configurations were considered. The first case study addressed a convergent–divergent nozzle governed by the two-dimensional compressible Euler equations. A parametric dataset was generated by varying the chamber-to-exit pressure ratio, leading to regimes ranging from fully subsonic solutions to mixed elliptic–hyperbolic behavior with shock formation in the divergent section. Within this framework, the GNN was trained to perform node-wise regression of Mach number and static pressure fields. The results showed that the network is able to reconstruct the global flow patterns across the investigated parametric range, capturing the transition between regimes and providing physically plausible solutions. Although the model is not intended to replace conventional solvers, the predicted fields can be exploited as physically consistent initial conditions for time-marching schemes, with the potential to reduce the number of iterations required to reach convergence, especially in challenging operating conditions. The second case study focused on a viscous incompressible flow over a backward-facing step, simulated in ANSYS by varying the Reynolds number. The same message-passing architecture was employed to predict the streamwise velocity component and the pressure distribution. The obtained results confirmed that the GNN can accurately reproduce the main bulk-flow features, including the downstream recovery and the large-scale effects of separation. However, the analysis also highlighted that matching primary fields through a standard mean-squared

error (MSE) loss does not necessarily guarantee an accurate prediction of near-wall gradients and derived wall quantities. In particular, even small absolute errors in the boundary-layer region can translate into large relative discrepancies in the wall-normal velocity gradient and therefore in the wall shear stress. To address this limitation, the training objective was augmented with a relative-error contribution. This modification preserved the quality of the global field reconstruction while significantly improving the agreement of near-wall behavior and wall-gradient trends, which are essential for viscous flows and friction-related metrics. Beyond the specific outcomes of each case study, several general observations can be drawn. First, the use of a single graph-based message-passing architecture proved to be flexible enough to model flows governed by different equations (Euler vs incompressible viscous flow) and characterized by different dominant physical mechanisms (shock waves vs boundary-layer and separation phenomena). Second, the work emphasized that loss-function design is a critical component of surrogate modeling. While an MSE formulation may be sufficient to achieve low global errors on smooth, high-magnitude fields, it can underweight physically critical regions where the solution magnitude is small but the sensitivity of derived quantities is high. Introducing relative-error terms represents an effective strategy to rebalance the training objective and improve physical consistency without sacrificing global accuracy. The present work also has limitations that open directions for future research. The datasets considered were restricted to a fixed geometry for each configuration and to a finite parametric range, and therefore generalization outside the training domain was not systematically addressed. Moreover, the surrogate models were trained in a purely data-driven manner, without enforcing conservation laws or boundary conditions as explicit constraints, which may become important when extending the approach to more complex flows or to broader operating ranges. Future work could build upon these limitations along several complementary directions. First, the prediction of integrated engineering quantities (e.g., reattachment length, skin-friction coefficient, and pressure losses) could be included as auxiliary targets in the training objective. Coupling node-wise field regression with such global metrics would directly penalize physically meaningful errors that may not be fully captured by pointwise losses, especially in near-wall regions where derived quantities depend on accurate gradients. A second important step is to quantify the practical computational benefit of the proposed approach in a solver-accelerated workflow. In particular for the compressible nozzle case, a dedicated assessment could compare the convergence behavior of conventional time-marching solvers when initialized with standard fields versus when initialized with GNN-predicted solutions. Measuring reductions in iteration count, runtime, and robustness

across operating regimes would provide a direct estimate of the real impact of the surrogate as a physically consistent initializer. In addition, operator-learning methods represent a promising alternative to standard surrogate regression. While graph neural networks can naturally process unstructured discretizations, it is still not fully understood how their predictions behave when the same physical configuration is represented by graphs of different sizes and resolutions. In other words, strong generalization across discretization levels and a clear notion of convergence under mesh refinement are not guaranteed in standard GNN surrogates. Neural operators, and in particular graph neural operators, aim to address this limitation by learning a discretization-consistent mapping between function spaces, so that increasing the graph resolution should yield predictions that converge toward the underlying continuous solution. Finally, the nozzle case motivates exploring models that not only provide high-quality initial conditions but also directly approach the converged steady solution with high fidelity. One possible strategy is to train the network as an accelerated time-advancement surrogate that learns an effective large time step, enabling coarse marching toward steady state. In practice, the model would approximate a macro-evolution operator over a much larger Δt than that typically used in explicit CFD time stepping, and the steady solution could be reached through a substantially small number of learned updates rather than the iterations often required by standard solvers. Such a learned macro-integrator could significantly reduce computational cost; however, its stability, robustness, and ability to preserve physically admissible solutions (especially in the presence of shocks) would need to be carefully assessed.

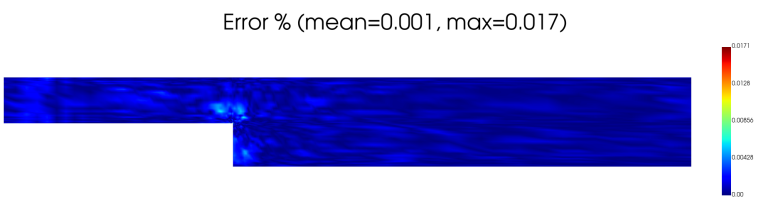
In conclusion, this thesis demonstrates that Graph Neural Networks constitute a promising and versatile surrogate modeling framework for fluid dynamics applications. When combined with an appropriate training objective and a careful evaluation of physically relevant quantities, the same message-passing architecture can capture distinct flow behaviors arising from different governing equations, supporting the broader applicability of GNN-based surrogates for accelerating simulation workflows and enabling rapid parametric analysis.

Appendix A

Pressure-field comparisons for the MSE-only model

(a) CFD target

(b) GNN prediction

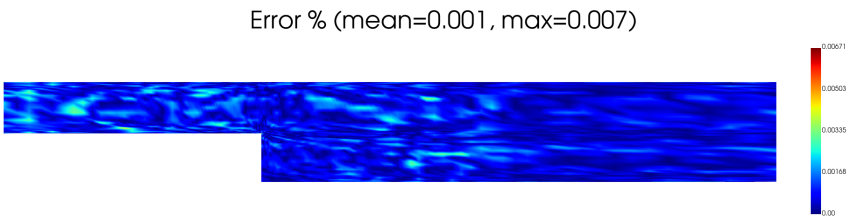


(c) Normalized absolute error (%)

Figure A.1: Pressure field for the validation case at $Re = 10$

(a) CFD target

(b) GNN prediction

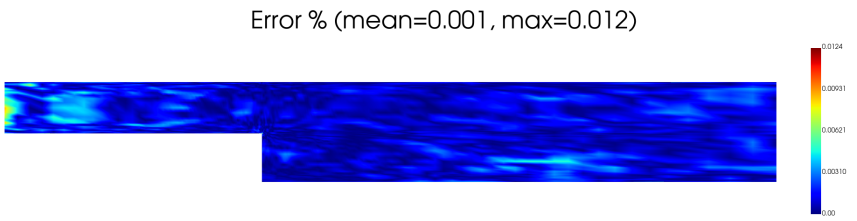


(c) Normalized absolute error (%)

Figure A.2: Pressure field for the validation case at $Re = 160$

(a) CFD target

(b) GNN prediction



(c) Normalized absolute error (%)

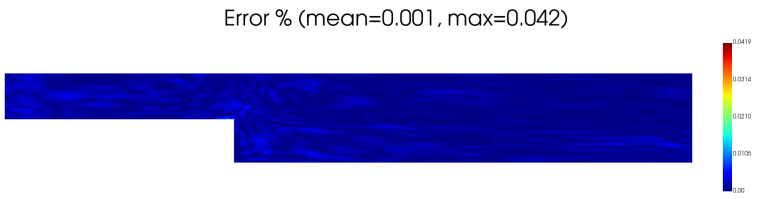
Figure A.3: Pressure field for the validation case at $Re = 300$

Appendix B

Pressure-field comparisons for the blended loss model

(a) CFD target

(b) GNN prediction

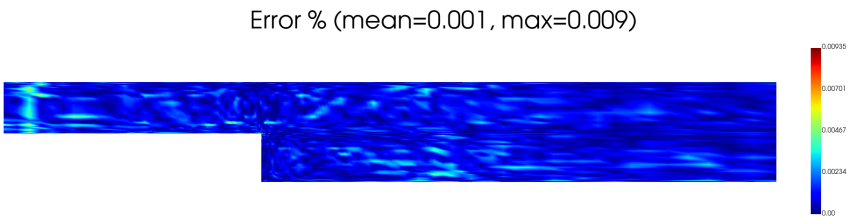


(c) Normalized absolute error (%)

Figure B.1: Pressure field for the validation case at $Re = 10$

(a) CFD target

(b) GNN prediction

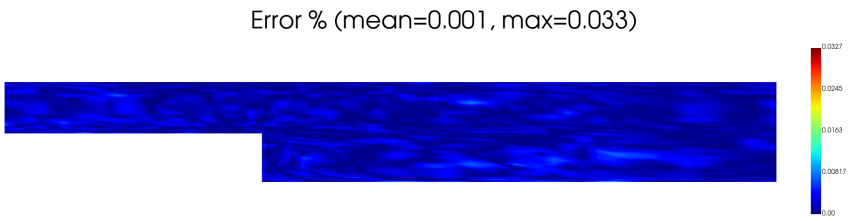


(c) Normalized absolute error (%)

Figure B.2: Pressure field for the validation case at $Re = 160$

(a) CFD target

(b) GNN prediction



(c) Normalized absolute error (%)

Figure B.3: Pressure field for the validation case at $Re = 300$