AI Mesh Informed Techniques for Optimising the Design Process

Swansea University
Prifysgol Abertawe

Callum Lock

College of Engineering

Swansea University

This dissertation is submitted for the degree of

Doctor of Philosophy

March 2025

Declaration

Statement 1

No part of this work has previously been submitted for any degree and is not being concurrently submitted in candidature for any degree at this or any other university.

Date: .... 20/08/2025 ....

Signature: ..... [redacted] ....(Candidate)

Statement 2

This thesis is the result of my own investigations, references to the work of others have been clearly acknowledged. A bibliography is appended.

Date: .... 20/08/2025 ....

Signature: ...... [redacted] ....(Candidate)

Statement 3

I give consent for the thesis, if accepted to be made available online in the University's Open Access Repository and for inter-library loan, and for the title and summary to be made available to outside organisations.

Date: .... 20/08/2025 ....

Signature: .... [redacted] ....(Candidate)

Abstract

This thesis presents a novel, data-driven framework for automatically generating near-optimal unstructured meshes for computational simulations. The primary objective is to reduce the manual effort and expert intervention typically required in mesh generation by leveraging historical simulation data and artificial neural networks (ANNs) to predict appropriate mesh spacing fields. The work is motivated by the growing availability of high-fidelity simulation data in industry and the need to streamline simulation workflows – particularly in the aerospace sector, where the mesh generation process remains one of the most resource-intensive steps.

Three different strategies are developed and evaluated. The first approach predicts the properties of point sources used to define the mesh resolution. The second introduces a coarse background mesh, onto which spacing functions are conservatively interpolated and predicted by ANNs. The third and final approach extends the method to fully anisotropic spacing by predicting the components of the metric tensor, allowing for directionally aligned mesh refinement. All three techniques are trained on datasets derived from prior simulations and are shown to generalise effectively to unseen geometric and flow conditions.

Extensive numerical experiments in three-dimensional compressible flow scenarios – including wings and full aircraft configurations, demonstrate that the proposed methods yield high-quality meshes capable of producing accurate solutions. Furthermore, an environmental impact analysis shows the potential for a substantial reduction in computational cost and energy usage, highlighting the ability of the methods outlined to be part of sustainable simulation practices.

This work lays the foundation for integrating machine learning into the meshing pipeline, enabling intelligent, scalable, and more efficient simulation-driven design across a wide range of engineering applications.

Keywords : Mesh generation; Machine learning; Near-optimal mesh prediction; Computational fluid dynamics

Acknowledgements

First of all, I would like to thank my supervisors, Professor Oubay Hassan, Professor Rubén Sevilla, and Dr Jason Jones, for giving me the opportunity to pursue research under their guidance. I am truly grateful for the invaluable support, encouragement, and insight they have provided throughout the course of this PhD. Their dedication to research and depth of knowledge has been a constant source of inspiration.

I would especially like to thank Professor Oubay Hassan. Over the years, he has taught me an incredible amount, ranging from mesh generation and solvers to the intricacies of Fortran 90. His door was always open, and no question was ever too small or too late to ask. I deeply appreciate his time, patience, and the confidence he placed in me. His steady support has been instrumental throughout this journey, and I consider myself incredibly fortunate to have worked under his supervision.

To Professor Rubén Sevilla, I am thankful for the clarity and structure he brought to my work. His ability to extract meaning from my sometimes disorganised thoughts and writing was remarkable, and his insistence on rigour helped me become a more precise and thoughtful person and researcher.

I would also like to thank Dr Jason Jones for all his support with the computational aspects of this work. Whether it was about running jobs, interpreting outputs, or simply understanding how things worked on the cluster, his advice was always appreciated.

To all three of you, thank you again. Your complementary strengths and consistent support have shaped not only this thesis but also the way I approach research and problem-solving. It has been a privilege to work under your supervision.

I would also like to thank Dr Xi, who joined partway through my PhD and, for a time, sat opposite me in the office. Over those months – and well beyond – he became both a generous colleague and a friend. His generosity with his time and his wealth of knowledge have had a profound impact on my understanding of so many areas. I am deeply grateful for the many discussions, insights, and patient explanations he shared throughout my PhD.

To my friends of A130, thank you for being a constant source of support, humour, and

motivation. Whether it was late nights in the office, weekends spent working, or the weekly pub quiz (which we never quite managed to win...), your presence made the hardest days manageable and the best days unforgettable. I am all the richer for having met each and every one of you, and I am truly grateful to have shared this chapter of my life with such an incredible group of people.

Finally, and most importantly, I would like to thank my family – specifically my mother, my step-father, and my nan. Your unwavering love, encouragement, and belief in me have been the foundation of everything I have accomplished. This thesis is not only the result of four years of research, but the culmination of over two and a half decades of your guidance, support, and quiet sacrifices. I would not be where I am today without you, and this achievement is every bit as much yours as it is mine.

I would also like to express my thanks to Airbus Defence and Space for their collaboration and financial support. Their engagement and contributions throughout the project are deeply appreciated. I am also thankful to the Engineering and Physical Sciences Research Council (EPSRC), under grant number EP/T517987/1, for their partial funding of this research.

Research Output

Journal Publications

- **Lock, C.**, Hassan, O., Sevilla, R., and Jones, J. (2023). Meshing using neural networks for improving the efficiency of computer modelling. Engineering Computers, 39(6):3791–3820.

- **Lock, C.**, Hassan, O., Sevilla, R., and Jones, J. (2025). Anisotropic mesh spacing prediction using neural networks. Computer-Aided Design.

Book Chapters

- **Lock, C.**, Hassan, O., Sevilla, R., and Jones, J. (2024). Predicting the near-optimal mesh spacing for a simulation using machine learning. In Lecture Notes in Computational Science and Engineering, Lecture Notes in Computational Science and Engineering, pages 115–136. Springer Nature Switzerland, Cham.

Conference Presentations

- **C Lock**, O Hassan, R Sevilla, J Jones, *AI Mesh Informed Techniques for Optimising the Design Process*, UK Association for Computational Mechanics (UKACM), Nottingham, United Kingdom, April 2022.

- **C Lock**, O Hassan, R Sevilla, J Jones, *Predicting the Near-Optimal Mesh Spacing for a Simulation Using Machine Learning*, SIAM International Meshing Roundtable Workshop (IMR), Amsterdam, Netherlands, March 2023.

- **C Lock**, O Hassan, R Sevilla, J Jones, *Predicting near-optimal meshes for CFD simulations*, IACM Computational Fluids Conference (CFC), Cannes, France, April 2023.

- **C Lock**, O Hassan, R Sevilla, J Jones, *AI Mesh-Informed Techniques for Optimising the Design Process*, UK Association for Computational Mechanics (UKACM), Warwick, United Kingdom, April 2023.

- **C Lock**, O Hassan, R Sevilla, J Jones, *AI Mesh-Informed Techniques for Optimising the Design Process*, UK Association for Computational Mechanics (UKACM), Durham, United Kingdom, April 2024.

- **C Lock**, O Hassan, R Sevilla, J Jones, *AI Mesh-Informed Techniques for Optimising the Design Process*, UK Fluids Conference, Swansea, United Kingdom, September 2024.

- C Lock, O Hassan, **R Sevilla**, J Jones, *Near-Optimal Mesh Generation Using Green AI*, Digital Twins in Engineering Conference (DTE) and Artificial Intelligence and Computational Methods in Applied Science (AICOMAS), Paris, France, February 2025.

- C Lock, O Hassan, **R Sevilla**, J Jones, *Near-Optimal Mesh Generation Using Green AI*, UK Association for Computational Mechanics (UKACM), London, United Kingdom, April 2025.

Awards

- **Best Student Paper**, IMR 2023 – *Predicting the Near-Optimal Mesh Spacing for a Simulation Using Machine Learning*

- **Best Session Presentation Award**, Swansea PGR Conference 2023

- **Laura Annie Wilson Prize**, UKACM 2024 – Awarded for the best presentation by a PhD researcher

Table of contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Computational simulations have become a cornerstone of modern engineering design. Across industries – from aerospace to automotive – high-fidelity simulations empower engineers to assess performance under a broad spectrum of operating conditions and systematically explore design alternatives with precision. Thereby reducing the dependence on physical prototyping and accelerating development timelines. As a result, simulation-driven design enables rapid, data-informed decision-making, facilitating efficient evaluation of geometric modifications and operational scenarios.

The ability to analyse complex flow phenomena through simulation is particularly vital in the aerospace sector, where small changes in geometry or operating conditions can lead to significant shifts in aerodynamic behaviour. Simulation-based optimisation allows engineers to quantify these effects and refine designs accordingly, leading to robust, high-performance solutions tailored to a range of flight conditions.

Central to this capability is Computational Fluid Dynamics (CFD), which plays a foundational role in the analysis and optimisation of systems involving fluid flow. CFD enables detailed prediction of flow behaviour around components of interest – span-

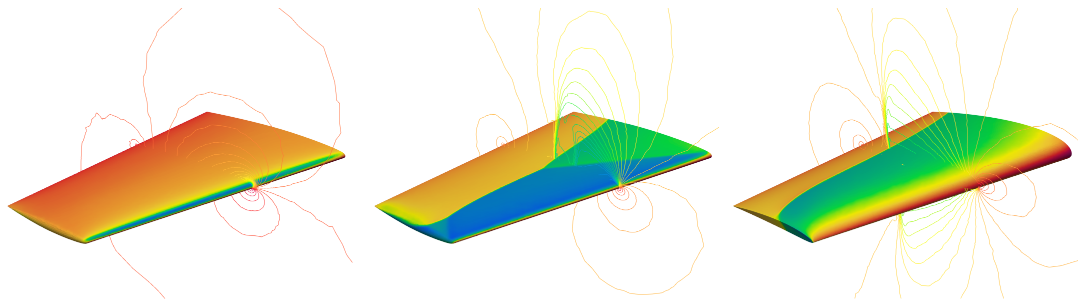


Figure 1.1: Flow solution variations over a transonic wing under different geometric and flow conditions. Surface colour contours represent the pressure coefficient (C_p). While the definition of C_p is introduced in Section 2.1.1, the key feature illustrated here is the emergence of sharp pressure discontinuities under varying configurations.

ning aircraft and automobiles to turbines, buildings, and medical devices – and supports iterative refinement of shapes, configurations, and performance. In commercial aviation, for instance, CFD has been instrumental in the development of winglets, the optimisation of lift-to-drag ratios, and substantial gains in fuel efficiency and emissions reduction.

Moreover, in the context of aerospace, CFD provides critical insights into flow regimes such as transonic, supersonic, and hypersonic flight. For aircraft operating near the speed of sound, transonic effects, such as shock wave formation, can have a pronounced impact on aerodynamic performance. These localised phenomena introduce sharp pressure gradients that elevate drag and influence stability. Through targeted geometric refinement, CFD enables accurate prediction and control of such effects, enhancing aircraft efficiency.

Figure 1.1 illustrates how modest changes in geometry or flow conditions can drastically alter flow structures. The first case shows a smooth, subsonic solution over a basic airfoil. In contrast, the second reveals the appearance of a lambda shock pattern under transonic conditions. The third demonstrates how modifying the airfoil alters the shock structure, emphasising the non-linear and highly sensitive nature of transonic flow.

Note that in this context, the absolute values of C_p are not the focus. Rather, this figure – and others like it throughout the thesis, serve to illustrate flow features such as gradients, discontinuities, and shock positioning. These patterns are central to the

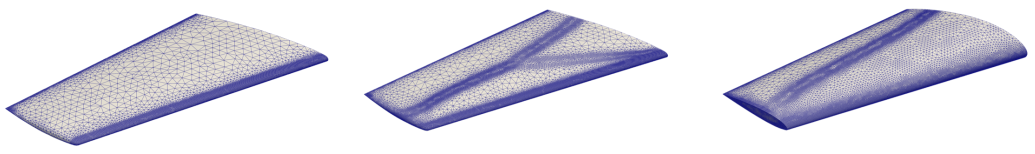


Figure 1.2: Near-optimal surface meshes corresponding to the flow cases in Figure 1.1, tailored to capture the key aerodynamic features of each configuration.

discussion, particularly in the context of mesh generation, where the structure of the field is more critical than the specific field magnitudes.

Accurately capturing such complex behaviours in numerical simulations depends critically on mesh generation – the process of discretising the computational domain into elements. Regions with steep gradients, shocks, or high curvature demand fine resolution to minimise numerical errors. For instance, lambda shocks must be sharply resolved, and curved regions such as leading edges require finer discretisation to model rapid changes in flow properties accurately.

Two general strategies are commonly adopted in mesh generation. The first is the use of uniformly heavily refined meshes designed to resolve all potential flow features from the outset. Although effective, this 'blanket refinement' approach is computationally expensive and environmentally unsustainable. It results in unnecessary resolution in smooth-flow regions, leading to inflated computational costs and energy consumption.

A more efficient alternative is the use of near-optimal meshes – refined to match the underlying flow features. These meshes concentrate elements in critical areas, such as near shocks or regions of curvature, while maintaining coarser resolution elsewhere. Figure 1.2 demonstrates this strategy: each mesh is tailored to its corresponding flow field, striking a balance between accuracy and computational efficiency.

Despite its advantages, generating near-optimal meshes remains a significant challenge. In manual mesh refinement, a skilled engineer typically specifies the mesh sizing locally, based on prior experience and understanding of the expected flow behaviour. Identifying regions that require high resolution – such as shocks, flow separation, or strong curvature – can be particularly difficult, especially in complex geometries or under varying operating conditions. As a result, conservative over-refinement is often employed to ensure accuracy, leading to increased computational cost and inefficiency.

Mesh adaptivity offers a promising alternative by iteratively refining the mesh based on error indicators derived from the computed solution. While effective, its success depends heavily on the quality of the initial mesh. If key features such as shocks are not adequately resolved early on, subsequent refinement may fail to capture them. This limitation is especially critical in transonic simulations, where subtle but important flow structures must be resolved from the outset.

To address these challenges, hybrid approaches are often employed, combining engineer-guided refinement with automated adaptivity. However, such techniques remain inherently reactive – requiring a solution to be computed before the mesh can be adapted.

Despite this, unstructured mesh generation continues to be a critical and often time-consuming step in simulation workflows, with a direct impact on both computational efficiency and solution accuracy. Conventional approaches frequently rely on heuristic refinement criteria and expert intervention, making them difficult to automate and scale (Dawes et al., 2001; Slotnick et al., 2014; Karman et al., 2017). This challenge is further exacerbated in design and optimisation studies, where multiple simulations are required across a range of operating conditions or geometric configurations. For complex geometries, generating a suitable unstructured mesh remains one of the most resource-intensive stages of the simulation process.

Meanwhile, the aerospace industry has accumulated vast datasets from thousands of CFD studies. These contain valuable insights into effective meshing strategies and the flow conditions they were intended to capture. Yet much of this historical knowledge remains underutilised.

Recent advances in artificial intelligence (AI) – and neural networks (NNs), in particular – offer new opportunities to leverage this data. In this thesis, the term neural network refers exclusively to artificial neural networks (ANNs): computational models inspired by biological neurons, designed to approximate complex, non-linear relationships between input parameters and output predictions. By training neural networks on past results, it becomes possible to predict mesh spacing requirements for new configurations without first solving the flow. These AI-driven tools can anticipate where fine resolution is needed, reducing over-refinement and accelerating the simulation setup process. This enables engineers to generate near-optimal meshes more efficiently, reducing both computational cost and environmental impact.

This thesis explores the development of such a strategy, leveraging neural networks to predict near-optimal mesh spacing fields for aerospace CFD applications. By integrating prior simulation data and error estimation techniques, it aims to provide a practical framework for efficient and accurate mesh generation across a wide range of configurations.

1.2 Background

The traditional workflow for simulation-based engineering design is illustrated in Figure 1.3. This process ensures that complex physical phenomena are accurately captured, beginning with the definition of operating conditions and the creation of a watertight CAD model. The operating conditions define the physical environment in which the system will operate, while the CAD model serves as the geometric foundation for simulation. Ensuring the CAD model is watertight (free of gaps or geometric inconsistencies) is essential for the subsequent mesh generation stage.

In the context of aerospace CFD applications, these operating conditions often reflect specific flow regimes (e.g., subsonic or transonic) and include additional parameters such as angle of attack or Reynolds number. The CAD geometry may represent individual components, such as an airfoil, a wing, or a full aircraft configuration, and must preserve all aerodynamic features critical to accurate flow prediction.

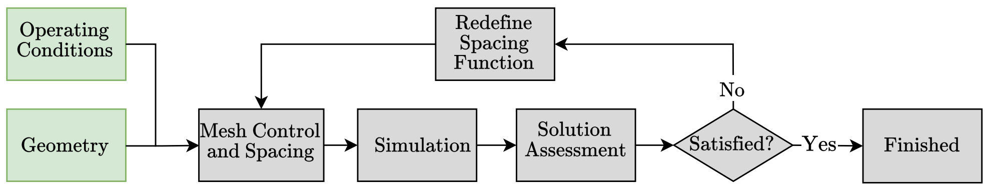


Figure 1.3: Traditional simulation-based engineering workflow, showing the iterative nature of mesh refinement in CFD.

Once the CAD model and flow conditions are defined, the workflow advances to mesh generation and control. This step involves discretising the domain into finite elements, forming the computational mesh. The resolution and distribution of mesh elements are guided by the flow conditions and geometric complexity. For example, regions prone to shock formation, boundary layers, or high curvature require local refinement to capture steep gradients and complex flow interactions. Mesh control can be achieved through manual specification or automated adaptive techniques, aiming to balance accuracy with computational cost by refining only in regions where fine resolution is needed.

In the CFD solving stage, the mesh is used to discretise and solve the governing equations of fluid dynamics, typically derived from the Navier–Stokes or Euler equations. The choice of numerical method – whether finite volume, finite element, or another formulation – does not alter the primary goal: to approximate the flow field accurately across the domain. Solvers iterate over the mesh, computing flow variables such as pressure, velocity, and temperature until convergence is reached. The fidelity of these results depends heavily on mesh quality, resolution, and alignment with flow features.

Once a solution is obtained, the final stage involves assessing its accuracy. This is typically done through error estimation techniques, residual monitoring, and convergence checks on integral quantities of interest – such as the lift and drag coefficients. If key flow features are under-resolved, or convergence has not been achieved, the process loops back to the mesh generation stage, where the spacing function is adjusted and a new mesh is generated. This iterative refinement continues until the solution is deemed mesh-independent, indicating that further refinement does not significantly alter the predicted quantities.

While effective, this iterative approach is time-consuming, often requiring significant engineering experience to predict where refinement is needed. Moreover, over-refinement in smooth-flow regions leads to wasted computational resources and longer simulation times. The challenge lies in generating meshes that are both accurate and efficient – refining only where necessary, and coarse elsewhere.

Despite the promise of adaptive methods and the wealth of accumulated simulation data, the mesh generation process remains a bottleneck in many high-fidelity CFD workflows. In recent years, AI has emerged as a powerful tool across scientific computing, offering the potential to automate traditionally expert-driven processes.

Within computational engineering, the use of ML has grown rapidly, with a strong emphasis on learning to predict physical phenomena such as flow fields, pressure distributions, or structural responses (Pfaff et al., 2020; Balla et al., 2021; Cai et al., 2022). These models often serve as surrogates or accelerators, reducing computational cost by approximating the output of expensive simulations.

In contrast, the use of ML to assist mesh generation has received comparatively less attention. Nonetheless, there is a growing body of work that explores this space. Early efforts date back to the 1990s, particularly within magnetic device simulations, where neural networks were trained to predict mesh density distributions (Dyck et al., 1992; Chedid and Najjar, 1996; Alfonzetti et al., 1996). More recently, approaches such as MeshingNet (Zhang et al., 2020, 2021) have employed neural networks to predict mesh spacing based on geometric information, boundary conditions, and parameters of the governing equations. Other works have explored neural network–driven adaptive strategies, where spacing is iteratively refined using output error indicators or reinforcement learning (Chen and Fidkowski, 2020; Bohn and Feischl, 2021; Yang et al., 2022). Neural networks have also been used to assess mesh quality (Chen et al., 2021), and in some cases, to inform adaptation in transient simulations (Manevitz et al., 2005).

These studies demonstrate the potential of machine learning to transform the mesh generation process from a reactive, expert-led task into a predictive, data-driven one.

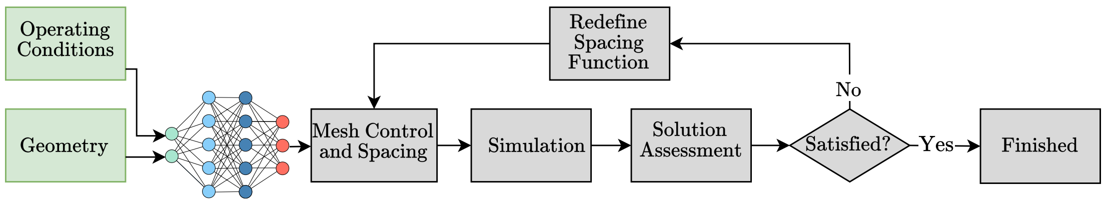


Figure 1.4: Proposed simulation workflow integrating AI-driven mesh spacing prediction.

However, most existing methods focus either on simplified problems or require extensive hand-tuning and domain-specific adaptation. This thesis builds on these ideas by proposing a generalisable framework that leverages solution fields from prior CFD simulations to predict near-optimal spacing distributions using neural networks. The goal is to automate the generation of tailored spacing functions -— independent of the underlying solver or geometry, thereby reducing setup time, improving efficiency, and expanding the applicability of high-fidelity CFD across the aerospace industry.

1.3 Objectives of This Thesis

This thesis aims to integrate AI into the simulation workflow for steady-state flow analysis by developing a framework that enables the automatic generation of near-optimal meshes based on input geometry and flow conditions. The focus is explicitly on the solution of steady-state problems, where the flow field does not evolve in time, and simulation efficiency and convergence behaviour are strongly tied to mesh quality. The objective is to replace the manual or iterative definition of mesh refinement with an AI-driven prediction of the spacing field, thereby accelerating the overall process and reducing reliance on domain-specific meshing expertise. The proposed workflow, illustrated in Figure 1.4, introduces a neural network to predict mesh spacing directly from geometry and operating conditions.

Specifically, the approach focuses on training neural networks (NNs) to predict local mesh spacing parameters informed by historical CFD data. The emphasis is placed on steady-state solutions of inviscid compressible flow problems, where accurate spatial resolution is critical for capturing features such as shocks. By leveraging large datasets of prior simulations—including geometries, operating conditions, meshes, and resulting flow fields, the AI system learns the underlying relationship between problem setup and ideal mesh distribution. This allows the network to generalise to unseen cases and generate high-quality, case-specific spacing fields without requiring a flow solution beforehand.

Incorporating AI into the meshing process offers several key benefits. First, it significantly reduces the time and effort required to produce a high-quality mesh. Second, by learning from previously validated cases, the AI ensures that its predictions are rooted in proven engineering practices. Third, the resulting meshes are more efficient – targeting resolution where it matters most, leading to faster simulations and lower computational costs.

Ultimately, this work presents a framework for intelligent, data-driven mesh generation. It demonstrates how the integration of neural networks with classical meshing techniques can enhance both the efficiency and accuracy of CFD workflows – while capturing and preserving the wealth of engineering knowledge embedded in historical simulation data.

1.4 Thesis Outline

This thesis is organised into eight chapters, each addressing a distinct aspect of the work. Chapters Two to Four provide the necessary background and theoretical context, with brief literature reviews and discussions of alternative methodologies included where relevant. Chapters Five to Seven introduce a series of novel methodologies for data-driven mesh generation, each of which is subsequently evaluated through numerical examples. Chapter Eight concludes the thesis with a summary of key findings and suggestions for future work. The thesis is supported by a set of appendices, which provide additional detail, supporting methodologies, and extended results that complement the main chapters.

The remainder of the thesis is organised as follows:

- *Chapter Two : Numerical Solutions of the Euler Equations.* This chapter introduces the fundamental CFD formulations, with emphasis on the inviscid Euler equations. The governing equations are presented alongside the necessary boundary conditions for aerodynamic simulations. The finite volume method is introduced as the core discretisation technique, and the construction of the dual mesh is described. Finally, details of the solver, including artificial dissipation, the solution procedure, and multigrid acceleration, are provided to complete the numerical framework.

- *Chapter Three : Unstructured Mesh Generation.* The chapter introduces the geometric formulation used to represent the computational domain and proceeds to define mesh elements with anisotropic properties through key mesh parameters. Strategies for mesh control are then presented, including the use of sources and background meshes to dictate the spacing and orientation of elements. The mesh generation process is described in two stages: surface meshing using an advancing front method, and volume meshing via Delaunay triangulation with point insertion. Finally, post-processing techniques for mesh enhancement are detailed to improve element quality.

- *Chapter Four : Neural Networks.* This chapter introduces the neural network framework used throughout this work, beginning with its foundations in regression modelling. The structure and function of feed-forward networks are presented, including forward propagation, activation functions, and cost functions tailored to scalar and vector prediction. Optimisation strategies such as SGD and Adam are discussed, followed by the backpropagation algorithm for computing parameter gradients. The chapter also outlines data preparation techniques and evaluation metrics.

- *Chapter Five : Predicting the Near-Optimal Mesh Using Sources.* This chapter introduces a mesh prediction strategy based on the use of point sources to represent spacing requirements. Starting from high-fidelity solution data, point sources are generated using Hessian-based error estimation and grouped into a global set. A neural network is trained to predict the characteristics of these sources—location, spacing, and radius—based on geometric and flow parameters. The resulting spacing functions are used to generate near-optimal meshes for unseen cases. Numerical examples demonstrate the effectiveness of the approach, and the chapter concludes with an analysis of its efficiency and environmental impact.

- *Chapter Six : Predicting the Near-Optimal Mesh Using a Background Mesh.* This chapter presents a novel strategy for predicting mesh spacing fields on a background mesh using neural networks. A conservative interpolation technique is introduced to transfer spacing from fine solution meshes to a fixed background mesh, enabling consistent training data across cases. The predicted spacing fields are used to generate near-optimal meshes for new configurations. The approach is compared against the source-based method introduced in *Chapter Five*, with an analysis of prediction accuracy, training efficiency, and data requirements. Numerical examples are presented for a wing and full aircraft configuration, demonstrating the robustness and scalability of the background mesh strategy.

- *Chapter Seven : Predicting the Near-Optimal Anisotropic Mesh Using a Background Mesh.* This chapter extends the background mesh approach introduced in *Chapter Six* to predict fully anisotropic mesh spacing fields. The methodology enables the prediction of a metric tensor, allowing for anisotropic refinement aligned with flow features. A mesh morphing technique is introduced to accommodate geometric variation, enabling training across different configurations. A neural network is developed to predict the components of the metric tensor from geometric and flow parameters. The approach is assessed on three-dimensional inviscid compressible flow simulations, including the ONERA M6 wing and a full aircraft geometry with eleven geometric parameters.

- *Chapter Eight : Concluding Remarks*. This chapter presents the main conclusions of the work, touches upon the ethical considerations of using AI, and outlines potential directions for future research.

- *Appendix A : Error-Based Element Sizing*. This appendix details the error estimation techniques used to determine optimal element sizing. It introduces a one-dimensional error indicator, extends the concept to three dimensions using the Hessian matrix, and explains the process of recovering second derivatives from numerical solutions on unstructured meshes.

- *Appendix B : Line and Plane Fitting via Total Least-Squares*. This appendix presents the use of total least-squares techniques to fit lines and planes to three-dimensional point data. The approach is employed to identify directional trends in mesh source distributions, enabling the merging of point sources into line sources for more efficient mesh generation.

- *Appendix C : Neural Network Hyperparameter Tuning*. This appendix presents a series of targeted studies on the influence of neural network hyperparameters. It evaluates optimisation algorithms and activation functions in the context of mesh spacing prediction, concluding with recommended configurations based on accuracy and training stability.

Chapter 2

Numerical Solutions of the Euler Equations

The following chapter introduces the fundamental principles and methodologies underlying the computational fluid dynamics (CFD) framework adopted in this research. The governing equations that describe the conservation of mass, momentum, and energy within a fluid domain are central to the study of fluid flow. In this work, the Euler equations are used to model the fluid behaviour. These equations describe the conservation of mass, momentum, and energy within a control volume and form the mathematical basis for modelling aerodynamic behaviour in the absence of viscous effects. The accurate numerical solution of the governing equations is essential for modelling fluid dynamics in scenarios such as shock wave formation, supersonic flows, and external aerodynamics.

This chapter begins by presenting the Euler equations in their conservative form, along with other key considerations. The accompanying boundary conditions required for a well-posed problem are also discussed, as they play a vital role in ensuring accurate and stable solutions to the governing equations. Following this, the numerical techniques employed for solving the Euler equations are detailed. Specifically, the finite volume method (FVM) is introduced as the discretisation approach, transforming the governing partial differential equations into a system of algebraic equations suitable for numerical computation.

This chapter includes the use of a multigrid solver, a robust numerical technique designed to accelerate the convergence of iterative methods by solving the governing equations at multiple levels of grid resolution.

In summary, this chapter provides a overview of the mathematical and numerical foundation for CFD mesh generation and simulation. This understanding is crucial for contextualising the subsequent development and application of neural networks to predict meshes for CFD simulations, as presented in later chapters.

2.1 Governing Equations

The fluid flow problems examined in this work are described using the Euler equations – which describe the conservation of mass, momentum, and energy within an inviscid fluid. These equations are derived from the fundamental physical principles of fluid mechanics and form a system of hyperbolic partial differential equations (PDEs). In the absence of external volume forces, the strong form of the Euler equations in differential form within a d-dimensional Cartesian domain ($\Omega \subset \mathbb{R}^d$) with a boundary $\partial\Omega$ can be expressed as

$$\mathbf{U}_t + \nabla \cdot \mathbf{F} = \mathbf{0} \qquad \text{in } \Omega \times (0, T_f], \tag{2.1a}$$

$$\mathbf{U}(\mathbf{x}, 0) = \mathbf{U}_0 \qquad \text{in } \Omega, \tag{2.1b}$$

$$\mathbf{B}(\mathbf{U}, \mathbf{U}_\infty) = \mathbf{0} \qquad \text{on } \partial\Omega. \tag{2.1c}$$

Here, $\mathbf{B}$ is the generic flux used to specify the boundary conditions for inflow, outflow, and wall boundaries. The $\mathbf{U}$ represents the vector of conserved variables, and $\mathbf{F}$

represents the flux tensor, which are given by

$$\mathbf{U} = \begin{bmatrix} \rho \\ \rho \boldsymbol{u} \\ \rho \epsilon \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} \rho \boldsymbol{u}^T \\ \rho \boldsymbol{u} \otimes \boldsymbol{u} + p\mathbf{I} \\ (\rho \epsilon + p)\boldsymbol{u}^T \end{bmatrix}. \tag{2.2}$$

Here, ρ is the fluid density, $\boldsymbol{u}$ is the velocity column vector, and ϵ is the total specific total energy (the total energy per unit volume), p is the fluid pressure and $\mathbf{I}$ is the $d \times d$ identity matrix. These equations describe, respectively, the conservation of mass, momentum, and total energy.

The Euler equations, alone, require additional closure relations to form a complete and solvable system. To obtain a closed system of equations, the equation of state relating the internal energy to pressure and density is needed. This closure is provided by assuming the fluid behaves as a calorically perfect gas. This assumption allows the use of the ideal gas law, which relates the specific internal energy ϵ to pressure p, and density ρ as

$$p = \rho R T, \tag{2.3}$$

and

$$\epsilon = c_v T + \frac{1}{2}\|\boldsymbol{u}\|^2. \tag{2.4}$$

Here, R is the real gas constant, and c_v is the specific heat at constant volume. The two constants are interlinked by the following relations,

$$c_v = c_p - R \tag{2.5}$$

and

$$\gamma = c_p / c_v, \tag{2.6}$$

where γ is the ratio of specific heats – or the adiabatic index. For air, under standard conditions, the value is typically 1.4.

An important derived quantity is the local speed of sound, a, which characterises the propagation speed of pressure disturbances in a compressible medium. For a calorically perfect gas, it is given by $a = \sqrt{\gamma RT}$, showing that the speed of sound depends solely on the local thermodynamic state, specifically the temperature. The Mach number, M, is a dimensionless parameter defined as the ratio of the flow velocity $\|\boldsymbol{u}\|$ to the local speed of sound,

$$M = \frac{\|\boldsymbol{u}\|}{a}. \tag{2.7}$$

The Mach number plays a central role in compressible flow analysis, as it governs both the physical behaviour and classification of the flow regime. Subsonic flow occurs when the Mach number remains below one throughout the domain and is characterised by smooth and continuous streamlines. For slender aerodynamic bodies, subsonic freestream conditions typically correspond to $M_\infty < 0.8$. Transonic flows, by contrast, contain a mixture of subsonic and supersonic regions and typically arise when $0.8 < M_\infty < 1.2$ (Anderson, 2016). These flows often involve the formation of shock waves —- narrow regions across which flow properties such as pressure, density, and velocity change abruptly. A shock is mathematically described as a discontinuity in the flow field and poses significant challenges for both numerical simulation and meshing.

In this thesis, the computational examples focus on steady flows in the subsonic and transonic regimes, with subsonic freestream conditions ($M_\infty < 1$). These flow regimes are of particular relevance to modern aerospace applications, such as commercial aircraft operating near transonic speeds. They present unique challenges in mesh generation, adaptive refinement, and accurate solution capture, particularly in the presence of strong gradients and shock structures.

2.1.1 Aerodynamic Coefficient Calculations

Aerodynamic coefficients are non-dimensional quantities used to summarise the total aerodynamics of an object in a flow field and are used extensively in this work. These coefficients provide a convenient way to compare aerodynamic forces and moments across different flow conditions, geometries, or scales, enabling engineers to generalise results and apply them to various applications. By expressing forces and moments in a dimensionless form, they allow results to be generalised and applied to various scales or scenarios independent of specific flow parameters.

The primary aerodynamic coefficients considered are the coefficient of lift (C_L) and the coefficient of drag (C_D). These coefficients quantify the aerodynamic forces acting on a body by decomposing the surface stresses into components normal and tangential to the flow direction, corresponding to lift and drag forces, respectively.

For inviscid flows, the only force acting on the surface arises from the pressure distribution, as viscous effects are neglected. The lift and drag coefficients are expressed as

$$C_L = \frac{1}{q_\infty S_\Gamma} \int_\Gamma (-p\mathbf{n}^w \cdot \mathbf{n}^\infty) d\Gamma, \tag{2.8}$$

$$C_D = \frac{1}{q_\infty S_\Gamma} \int_\Gamma (-p\mathbf{n}^w \cdot \mathbf{t}^\infty) d\Gamma, \tag{2.9}$$

where $q_\infty = \frac{1}{2}\rho_\infty U_\infty^2$ is the freestream dynamic pressure, and S_Γ is the reference area used to normalise the aerodynamic forces. In aerospace applications, S_Γ typically corresponds to the platform area of the wings. The integrals are taken over the surface Γ, representing the wetted surface of the body (e.g. wing or full aircraft).

Here, $\mathbf{n}^w$ denotes the unit vector normal to the wall surface, pointing outward from the body. The vectors $\mathbf{n}^\infty$ and $\mathbf{t}^\infty$ represent the unit normal and tangential directions, respectively, of the freestream velocity vector. These are used to project the pressure forces into the lift and drag directions.

Lastly, in addition to the lift and drag coefficients, the coefficient of pressure C_p is a key aerodynamic parameter that describes the non-dimensional pressure distribution on the surface of the body,

$$C_p = \frac{p - p_\infty}{q_\infty}, \tag{2.10}$$

where p_∞ is the freestream pressure. The coefficient of pressure provides a direct measure of the pressure forces acting on the surface and is essential for understanding flow separation, stagnation points, and other critical flow phenomena.

2.2 Boundary Conditions

Boundary conditions are essential for solving the Euler equations (Equation 2.1), as they provide the necessary constraints to ensure a well-posed problem. The Euler equations describe the conservation of mass, momentum, and energy in a fluid; however, without appropriate boundary conditions, the solution to these equations would be indeterminate. Boundary conditions specify the behaviour of the fluid at the domain's boundaries, such as inflow, outflow, and walls, ensuring that the solution is physically realistic and unique. In aerodynamic applications, boundary conditions provide critical information about inflow, outflow, and wall interactions. They govern how mass, momentum, and energy enter or leave the domain, as well as how the fluid interacts with solid objects such as airfoils or fuselages.

2.2.1 Wall Boundary Condition

Wall boundary conditions describe the interaction of the fluid with a solid boundary, such as the surface of an aircraft wing or the walls of a wind tunnel. These boundaries are significant in aerodynamic simulations, as they directly influence the flow field, pressure distribution, and aerodynamic forces acting on the solid surface.

For inviscid flows governed by the Euler equations, the primary condition at a wall is the no-penetration condition, which states that the velocity of the fluid normal to the surface must be zero. Mathematically, this is expressed as

$$\boldsymbol{u} \cdot \boldsymbol{n}^{w} = 0, \tag{2.11}$$

where $\boldsymbol{u}$ is the fluid velocity vector and $\boldsymbol{n}$ is the unit normal vector to the wall face. In numerical simulations, the velocity component normal to the wall is set to zero, while the tangential velocity components remain unconstrained, allowing the fluid to slide along the surface. This sliding behaviour reflects the inviscid assumption of the Euler equations, which neglect viscous effects such as shear stress. This condition ensures that the fluid cannot pass through the wall, thereby enforcing the impermeability of the solid boundary.

2.2.2 Symmetry Boundary Condition

A symmetry boundary condition is a specific type of boundary condition used to exploit the inherent symmetrical properties of a problem, thereby reducing computational effort and complexity. When a physical system exhibits symmetry, the behaviour of the fluid on one side of a boundary can be mirrored across that boundary, meaning the flow properties (such as velocity, pressure, and temperature) remain unchanged when reflected across the plane of symmetry.

In practical terms, applying a symmetry boundary condition assumes no mass, momentum, or energy flux across the symmetry plane. This implies that the normal component of the velocity vector relative to the symmetry plane is zero, and the density and total energy derivatives with respect to the boundary normal are also zero. That is,

$$\boldsymbol{u} \cdot \hat{\mathbf{n}} = 0 \tag{2.12}$$

and

$$\frac{\partial \rho}{\partial \hat{\mathbf{n}}} = \frac{\partial \epsilon}{\partial \hat{\mathbf{n}}} = 0 \tag{2.13}$$

where $\hat{\mathbf{n}}$ is the symmetry boundary normal.

Essentially, the flow on one side of the plane is identical to the flow on the other side, allowing the simulation to focus only on one portion of the domain. This not only simplifies the computational domain but also significantly reduces the computational resources required to solve the problem – thus achieving efficiency without sacrificing accuracy.

2.2.3 Inflow and Outflow Boundary Condition

The numerical simulation of external flows is conducted within a finite computational domain. To accurately model the flow, artificial far-field boundary conditions are applied to satisfy two key requirements. First, the truncation of the computational domain must not significantly affect the solution, ensuring that the results closely approximate those of an infinite domain. Second, any disturbances leaving the domain must not be reflected back into the interior, as such reflections would introduce non-physical artefacts (Tamura and Fujii, 1993).

The selection of appropriate far-field boundary conditions relies on the concept of characteristic variables, which govern the behaviour of flow quantities at the boundaries (Ni, 1982). The far-field can be divided into two distinct regions: inflow, where fluid enters the computational domain, and outflow, where fluid exits the domain. Both regions require careful treatment to ensure that the numerical solution remains stable and consistent with the physical problem being modelled.

For an inviscid subsonic problem, the inflow boundary condition requires four conditions to define the system properly. In contrast, only the far-field pressure must be defined for the outflow boundary condition, with the remaining variables being extrapolated from the interior nodes.

2.3 Discretisation and Finite Volume Method

The governing equations of fluid dynamics, such as the Euler equations, are expressed as partial differential equations (PDEs). These equations describe the conservation of mass, momentum, and energy within a fluid domain and are generally continuous in both space and time. This means that the variables of interest are defined at every point in the domain and continuously vary. Solving these equations analytically for real-world problems is, however, often infeasible due to the complexity of the equations, the non-linearity of the flow physics, and the complex geometries.

To make the problem practically solvable, the domain and the governing equations must be discretised, transforming the continuous system into a discrete one. Discretisation replaces the continuous domain Ω with a finite number of subdomains, denoted as $\{\Omega_K \subset \Omega\}$, where $K = 1, 2, \ldots, N_{\text{el}}$, with N_{el} representing the total number of subdomains or elements. The computational domain Ω is divided into N_{el} non-overlapping elements, $\Omega_K \subset \Omega$,

$$\bigcup_{K=1}^{N_{\text{el}}} \Omega_K = \Omega, \quad \Omega_i \cap \Omega_j = \emptyset \quad \forall\, i \neq j. \tag{2.14}$$

Continuous variables, such as velocity and pressure, are replaced by their discrete counterparts, which are defined only at specific points within the finite subdomains. This process enables numerical methods to compute an approximate solution to the PDEs, making it possible to solve complex fluid flow problems.

The discretisation process is central to all numerical methods in CFD, including finite difference, finite element, and finite volume methods. Among these, the finite volume method (FVM) is particularly well-suited for solving conservation laws, such as the Euler equations, because it ensures the conservation of mass, momentum, and energy at the discrete level. The following subsections will detail the finite volume method and its application in this thesis.

2.3.1 Finite Volume Method

The finite volume method (FVM) is a widely used numerical scheme for solving partial differential equations, particularly those that describe conservation laws. It belongs to the family of weighted residual methods, where the governing equations are integrated over a set of subdomains to obtain a discretised formulation.

A general system of governing equations can be expressed in the form

$$\mathcal{L}_i(\mathbf{u}(\mathbf{x}), \mathbf{x}) = 0, \quad \mathbf{x} \in \Omega, \quad i \in [1, n], \tag{2.15}$$

where $\mathbf{u}(\mathbf{x})$ and $\mathbf{x}$ are the dependent and independent variables, respectively, $\mathcal{L}_i$ represents the differential operator acting on the dependent variables and n denotes the number of differential equations in the system. To obtain a numerical approximation, the equations are multiplied by a set of weighting functions, $W_K(\mathbf{x})$, and integrated over the discretised domain, Ω_d. Expressed as

$$\int_{\Omega_d} \mathcal{L}_i(\mathbf{u}(\mathbf{x}), \mathbf{x}) W_K(\mathbf{x})\, d\mathbf{x} = 0, \quad K \in [1, N], \tag{2.16}$$

where $W_K(\mathbf{x})$ are the weighting functions. To *solve* the differential equations is to find an approximate solution, $\tilde{u}$, that satisfies this system of equations whilst adhering to the boundary conditions of the problem.

In the finite volume method, the weighting functions are chosen to simplify the integration process and enforce local conservation properties. Specifically, the weighting functions, $W_K(\mathbf{x})$, are defined as

$$W_K(\mathbf{x}) = \begin{cases} 1, & \mathbf{x} \in \Omega_K, \\ 0, & \mathbf{x} \notin \Omega_K. \end{cases} \tag{2.17}$$

This choice of weighting function ensures that each control volume contributes inde-

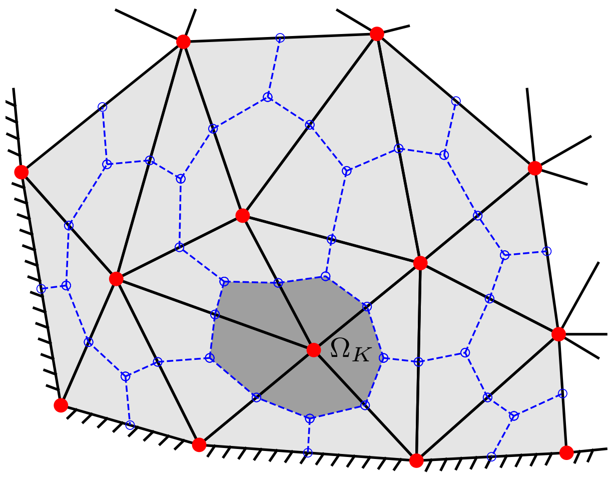


Figure 2.1: Illustration of the vertex-centred finite volume scheme for a given mesh, with the solid black lines representing the edges of the original mesh. The control volumes defined by the dual mesh, Ω_K, are outlined by the blue dashed lines, while the solid red circles indicate the locations of the unknowns.

pendently to the discretised equations, with no overlap or gaps between control volumes. The integration over each control volume ensures that the governing equations are satisfied in an average sense within each subdomain. This approach results in a system of discretised equations that approximate the original PDEs whilst conserving the physical quantities, mass, momentum, and energy at the discrete level.

Finite volume schemes can generally be classified into two main types: vertex-based and element-based schemes, with the distinguishing feature being the control volume, Ω_K. The solver used in this work employs a vertex-based scheme, where the control volumes are constructed around the vertices (or nodes) of the mesh, forming a dual mesh – its construction is outlined in Section 2.3.2. An illustration of the vertex-centred control–volume definition to FV for an unstructured triangular is shown in Figure 2.1.

The vertex-centred approach is favoured due to its computational efficiency and numerical robustness on unstructured tetrahedral meshes. A primary advantage lies in its memory efficiency, as the number of nodes in a three-dimensional mesh is significantly smaller than the number of elements – a ratio of approximately 1 to 5 (Giacomini et al., 2025; Hassan, 2025). Associating control volumes with nodes rather than elements reduces the degrees of freedom, minimising memory usage while retaining sufficient resolution – a crucial consideration for large-scale simulations.

The approach also simplifies interpolation and boundary condition enforcement, as the flow variables are stored at nodes, which inherently lie on domain boundaries. This property allows boundary conditions to be applied directly to nodes, eliminating the need for auxiliary reconstruction steps. For example, no-slip walls or symmetry planes are enforced straightforwardly, improving numerical stability.

2.3.2 Constructing the Dual Mesh

In the vertex-based method, the discrete dependent variables are defined at the nodes of the computational mesh. The control volumes are constructed around these nodes, ensuring that each control volume is uniquely associated with an individual node and does not overlap with others. The connecting surfaces between these control volumes form the dual mesh.

In three dimensions, the median dual mesh is constructed by connecting the midpoints of the edges, the centroids of the faces, and the centroids of the surrounding elements (Sørensen et al., 2003a). The dual mesh surfaces are defined using triangular facets, where each facet connects the midpoint of an edge, the centroid of a neighbouring element face, and the centroid of the adjacent element. Together, these triangular facets form the polygonal surfaces that encloses the control volume around a given node. The result is a non-overlapping, closed control volume that accurately partitions the computational domain. Figure 2.2 shows a portion of this dual mesh construction about a given node inside a tetrahedral element. Additionally, a complete dual around an internal node I is shown in Figure 2.3

This construction ensures that the control volumes are well-defined and suitable for integrating fluxes across their boundaries, enabling the finite volume method to maintain the conservation of mass, momentum, and energy throughout the domain.

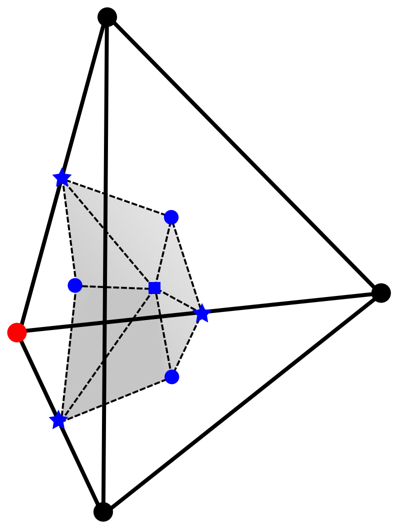


Figure 2.2: Illustration of a portion of the dual mesh, shown by the grey faces, constructed around a given node marked in red. The blue stars indicate the midpoints of the edges, the blue circle represents the face centroids, and the blue square marks the element centroid.

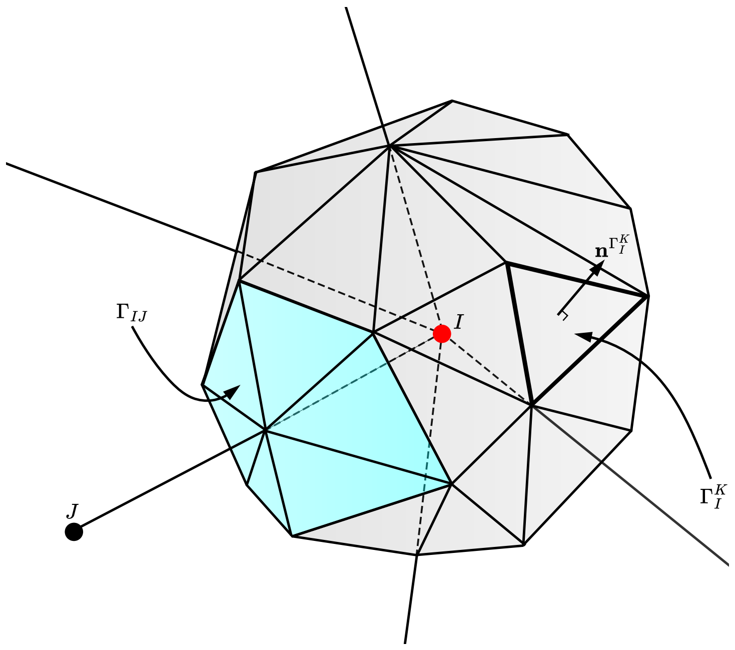


Figure 2.3: Illustration of the dual mesh surrounding an internal node I. The dual is constructed by a closed set of planar triangular facets Γ_I^K. Each facet only touches a single edge, and the set of facets touching the edge spanning between nodes I and J is termed Γ_{IJ}.

2.3.3 Discretised Euler Equation

To solve the Euler equations numerically, they must be discretised into a form suitable for computation. Multiplying the Euler equations by the weighting function previously outlined – Equation 2.17, and integrating over a control volume, Ω^I, transforms the differential form of the governing equations –Equation 2.1, into their integral – or weak form,

$$\int_{\Omega^I} \frac{\partial \mathbf{U}}{\partial t} d\Omega + \int_{\partial\Omega^I} \mathbf{F} \cdot \mathbf{n} d\Gamma = \mathbf{0}. \tag{2.18}$$

This formulation expresses the conservation laws in a manner well-suited for numerical methods, as it directly accounts for fluxes across the boundaries of control volumes. To obtain a discrete numerical solution, the surface and volume integrals present in the governing equations must be approximated. In the node-based scheme used, the fluxes are integrated over the control volumes of the dual mesh by looping over the edges of the original mesh.

The semi-discretised form of the Euler equation 2.18 is given by

$$\begin{aligned} |\Omega_I| \frac{d\mathbf{U}^I}{dt} + \sum_{J\in\Lambda_I} \frac{1}{2}\left(\mathbf{F}^I + \mathbf{F}^J\right) \cdot \mathbf{C}^{IJ} + \sum_{J\in\Lambda_I^B} \mathbf{F}^I \cdot \mathbf{D}^{IJ} \\ - \sum_{J\in\Lambda_I} m^{IJ}\alpha_{IJ}\left(\mathbf{E}^J - \mathbf{E}^I\right) - \sum_{J\in\Lambda_I} \epsilon_2 \alpha_{IJ} N^{IJ}\left(\mathbf{U}^J - \mathbf{U}^I\right) = \mathbf{0}, \end{aligned} \tag{2.19}$$

where Λ_I denotes the set of nodes connected to a given node I through an edge, and Λ_I^B denotes the set of nodes connected to node I by an edge on the computational boundary – applicable if node I is a boundary node. The first term of the equation represents the temporal discretisation, the second and third terms represent the discredited inviscid fluxes, and the fourth and fifth terms are the artificial dissipation terms.

2.3.3.1 Discretised Inviscid Terms

Regarding the discredited inviscid fluxes, the numerical integration of the flux over the dual mesh segment is associated with an edge connected to a given node, I. For the edges which are connected to an interior node – belonging to Λ_I, the flux is assumed to be constant, equal to the flux approximate value at the midpoint. The flux coefficient of an interior edge and a boundary edge, $\mathbf{C}^{IJ}$ and $\mathbf{D}^{IJ}$, are given by

$$\mathbf{C}^{IJ} = \sum_{K \in \Gamma_{IJ}} A_{\Gamma_I^K} \mathbf{n}^{\Gamma_I^K}, \tag{2.20}$$

$$\mathbf{D}^{IJ} = \sum_{K \in \Gamma_{IJ}^B} A_{\Gamma_I^K} \mathbf{n}^{\Gamma_I^K}, \tag{2.21}$$

where $A_{\Gamma_I^K}$ is the area of facet Γ_I^K and $\mathbf{n}^{\Gamma_I^K}$ is outward facing unit normal with respect to the control volume. This configuration is illustrated in Figure 2.3.

2.3.3.2 Artificial Dissipation

The central difference scheme used in this work is known to be inherently unstable when applied to equations of hyperbolic nature, such as the Euler equations. To stabilise the scheme and suppress non-physical oscillations arising from the convective terms, artificial dissipation is introduced (Hirsch, 2007). The stabalisation is provided by the fourth term of Equation 2.19, that corresponds to a third-order biharmonic dissipation term – based on JST scheme (Hirsch, 2007).

The scaled artificial dissipation flux $\mathbf{E}(\mathbf{U})$ at node I takes the form

$$\mathbf{E}^I = \frac{1}{\sum_{K \in \Lambda_I} l_{IK}^{-1}} \sum_{K \in \Lambda_I} l_{IK}^{-1} (\mathbf{U}^K - \mathbf{U}^I),$$

where l_{IK} is the length of the edge connecting node I and node K. This scaling ensures that the artificial dissipation behaves similarly to a discrete fourth-order difference scheme. The coefficients m and α of the biharmonic dissipation term are given by

$$m^{IJ} = \max(0, \epsilon_4 - \epsilon_2 N^{IJ}) \tag{2.22}$$

$$\alpha_{IJ} = \frac{1}{|\Lambda_I| + |\Lambda_J|} \min(\frac{|\Omega_I|}{\Delta\tau_I}, \frac{|\Omega_J|}{\Delta\tau_J}), \tag{2.23}$$

where ϵ_2 and ϵ_4 are user-defined dissipation factors, taken to be 0.3 and 0.2 in this work. These factors control the strength of the dissipation, allowing it to be tuned based on the problem at hand. The $|\Lambda_J|$ represents the number of edges connected to node I, Ω_I is the volume of the control volume, and $\Delta\tau$ is the local time-step.

However, the biharmonic term is ineffective at damping oscillations in regions with high gradients – such as shocks, necessitating the localised addition of the harmonic dissipation – the fifth term of Equation 2.19. Its contribution is primarily significant in areas of steep pressure gradients due to the inclusion of a pressure switch N^{IJ}, defined by

$$N^{IJ} = \max(|\Delta p_I|, |\Delta p_J|) \tag{2.24}$$

with

$$\Delta p_I = 12 \frac{\sum_{K\in\Lambda_I}(p_K - p_I)}{\sum_{K\in\Lambda_I}(p_K + p_I)} \tag{2.25}$$

where p_I represents the pressure at an arbitrary node I. These pressure sensors are triggered in regions with strong gradients, such as shocks within the flow, ensuring that the scheme effectively stabilises the solution in these critical areas. As a consequence of equations 2.22 and 2.24, when the harmonic term is strong, it adds enough stabilisation so that the biharmonic term is not needed.

2.4 Solution procedure

The discretised form of the Euler equations in Equation 2.19, can be expressed as

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{R}^I = \mathbf{0}, \tag{2.26}$$

where $\mathbf{R}^I$ represents the residual of the governing equations at a given node,

$$\begin{aligned}\mathbf{R}^I \equiv \frac{1}{|\Omega^I|}\Big[\sum_{J\in\Lambda_I} \frac{1}{2}\left(\mathbf{F}^I + \mathbf{F}^J\right)\cdot \mathbf{C}^{IJ} + \sum_{J\in\Lambda_I^B} \mathbf{F}^I \cdot \mathbf{D}^{IJ} \\ - \sum_{J\in\Lambda_I} m^{IJ}\alpha_{IJ}\left(\mathbf{E}^J - \mathbf{E}^I\right) - \sum_{J\in\Lambda_I} \epsilon_2 \alpha_{IJ} N^{IJ}\left(\mathbf{U}^J - \mathbf{U}^I\right)\Big],\end{aligned} \tag{2.27}$$

and t is a pseudo-time variable introduced to iteratively advance the solution towards a steady-state solution. For steady-state problems, the objective is to drive the residual $\mathbf{R}^I$ to zero, ensuring that the governing equations are satisfied at every discrete point in the domain. This is achieved by employing an iterative procedure that updates the solution until convergence criteria are met.

Globally, the total number of unknowns in the system can be extremely large, reaching tens of millions or more. Consequently, solving the discretised governing equations poses a significant challenge, often demanding substantial computational resources and lengthy solution times.

With the discrete equations formulated, the unknown variables must be resolved using an appropriate solution procedure. In general, the procedure for determining the solution can be classified into two categories: implicit and explicit schemes. Each of these approaches offers distinct advantages and challenges depending on the size of the computational domain and the complexity of the problem.

Implicit schemes solve systems of equations where the unknowns are coupled through the discretised governing equations. This coupling arises from the interaction of unknowns with their neighbours due to flux terms and spatial derivatives. The resulting system is represented as a large, sparse linear system of equations, requiring matrix inversion for a solution. Although implicit methods are robust and converge with fewer iterations, they impose significant computational costs. For large-scale problems involving meshes with millions of unknowns, the matrix inversion process can be computationally intensive, demanding substantial storage and runtime, especially when direct or iterative solution techniques are employed.

Explicit schemes, in contrast, do not require the storage or inversion of large matrices. Instead, the updated solution is directly computed as a function of known local quantities. This simplicity comes at the cost of slower convergence, as explicit schemes generally require many more iterations than implicit schemes to reach a converged solution. However, explicit methods have two advantages: they require considerably less storage and are inherently well-suited for parallelisation.

Among the many methods available in the literature for obtaining explicit solutions, the solver employs the multi-stage Runge-Kutta time-stepping scheme describe. The three-stage Runge-Kutta method, used to solve the ordinary differential equation is

$$\begin{aligned}
\mathbf{U}^{I,s_0} &= \mathbf{U}^{I,n}, \\
\mathbf{U}^{I,s_1} &= \mathbf{U}^{I,s_0} - \alpha_1 \, \mathrm{CFL} \, \Delta t \, \mathbf{R}(\mathbf{U}^{s_0}), \\
\mathbf{U}^{I,s_2} &= \mathbf{U}^{I,s_0} - \alpha_2 \, \mathrm{CFL} \, \Delta t \, \mathbf{R}(\mathbf{U}^{s_1}), \\
\mathbf{U}^{I,n+1} &= \mathbf{U}^{I,s_0} - \alpha_3 \, \mathrm{CFL} \, \Delta t \, \mathbf{R}(\mathbf{U}^{s_2}),
\end{aligned} \tag{2.28}$$

where $\mathbf{U}^{I,n}$ represents the unknowns at iteration n, and $\mathbf{U}^{I,s_x}$ corresponds to the unknowns at Runge-Kutta iteration s. The local psuedo-time-step is denoted by Δt, and the coefficients α_x are selected as 0.6, 0.6, and 1.0, respectively (Sørensen et al., 2003c). The maximum allowable CFL number for stability is dependent on the number of stages in the scheme, $P - 1 = 2$ (Swanson and Turkel, 1997).

2.4.1 Multigrid

The complex interactions of flow structures result in the solutions of fluid flow problems are often characterised by a wide range of length scales. Considering the flow structure around an aircraft, near the aircraft geometry, localised disturbances – like boundary layers and stagnation points, are expected. Additionally, these structures considerably affect the fluid flow afar, resulting in features such as wakes and shocks. This means that some regions of the flow are governed by local influences while others are significantly affected by distant flow patterns.

In numerical systems, this complexity translates into strong couplings between unknowns that are weakly connected in the discretisation. Explicit schemes, while efficient at reducing high-frequency local errors, struggle to propagate information across the domain to eliminate low-frequency, global errors. As a result, explicit solvers require many iterations to address these global inaccuracies, leading to slower overall convergence.

Multigrid methods are specifically designed to reduce this issue and subsequently accelerate convergence by simultaneously dampening errors across all frequencies while retaining the low computational cost and memory efficiency of explicit schemes. These methods achieve this by using multiple levels of grid resolution. Two main approaches to multigrid exist: the geometric multigrid, which operates directly on multiple discretisation levels, and the algebraic multigrid, which works on the linear system of equations.

The algebraic multigrid approach involves linearising the non-linear system of equations and generating smaller, successively coarser matrix systems. This effectively increases the domain of influence of each mesh node, allowing information to propagate across a larger portion of the computational domain and improving convergence rates. However, the method requires the computation and storage of the Jacobian matrix, which can be computationally expensive and memory-intensive. This limitation becomes particularly unfavourable when solving large systems of equations, making the algebraic approach less practical for large-scale CFD simulations compared to geometric multigrid methods (Stüben, 2001).

Alternatively, geometric multigrid involves solving the governing equations on a hierarchy of grids with varying coarseness. Each grid level targets a specific range of error frequencies, with finer grids addressing high-frequency errors and coarser grids focusing on low-frequency errors. On coarse grids, where the domain of influence for a single node is larger, the low-frequency errors from the fine grid are transformed into high-frequency errors that are more easily eliminated (Wesseling and Oosterlee, 2001).

This allows explicit relaxation schemes to effectively reduce global errors on coarse grids at a significantly lower computational cost due to the reduced number of nodes and larger allowable (pseudo) time-steps. As a result, coarse grids are highly efficient in smoothing low-frequency errors, enabling rapid convergence on the finest grid.

The solver used in this work employs a geometric multigrid approach using the Full Approximation Storage (FAS) scheme to accelerate convergence for non-linear problems (Sørensen et al., 2003c; Swanson and Turkel, 1997; Sørensen et al., 2003a). The FAS scheme extends geometric multigrid to handle non-linear equations directly, rather than relying on linearisation. At each grid level, FAS solves the full non-linear problem and transfers both the approximate solution and the residual to coarser grids. This allows corrections from the coarser grids to improve the fine-grid solution iteratively while preserving the non-linearity of the equations. By combining non-linear solving capabilities with efficient error smoothing across multiple grids, FAS is well suited for solving complex, non-linear CFD problems like those encountered in this work.

In this work, the coarser grid levels are not generated through re-meshing; instead, they are obtained via agglomeration of the control volumes from the finer grid. This agglomeration process combines adjacent control volumes to form larger cells, effectively coarsening the mesh while preserving the underlying structure of the initial fine grid. A three-level multigrid structure is employed in which each level corresponds to a distinct mesh: the original fine mesh and two successively coarser meshes constructed via agglomeration. This hierarchical arrangement accelerates convergence whilst maintaining consistency with the discretisation used in the finest grid. Further details of the agglomeration methodology can be found in (Sørensen et al., 2003b).

Chapter 3

Unstructured Mesh Generation

In computational fluid dynamics (CFD), the accuracy and efficiency of a simulation depend heavily on how the computational domain is discretised. In the previous chapter, the finite volume method (FVM) was introduced as the method for solving the discretised governing equations, which requires the domain to be partitioned into control volumes. This chapter focuses on the critical step of mesh generation, where the physical domain is discretised into elements that form the basis of the numerical solution.

Mesh generation is a fundamental process that converts the geometry of the problem into a discrete representation, allowing for the accurate integration of the governing equations across the domain. The mesh defines where flow variables, such as velocity and pressure, are computed and how information propagates between nodes. The quality of the mesh directly influences the stability, convergence, and accuracy of the CFD solution, making it essential to construct a mesh that adequately resolves key flow features while minimising computational cost.

The chapter begins with a discussion on geometry modelling, where the curves and surfaces that define the problem domain are described. Next, the process of mesh generation is detailed, starting with the definition of elements and how mesh control techniques, such as background meshes and sources, guide the distribution of elements across the domain. The discretisation of curves, surfaces, and volumes is then examined to ensure the mesh conforms to the geometry and properly captures critical flow regions. Additional sections cover the mesh enhancements necessary to refine and smooth the mesh prior to simulations.

3.1 Geometry Modelling

Before mesh generation can begin, the boundary of the computational domain must be precisely defined and represented in a form suitable for discretisation. This boundary serves as the interface between the physical geometry and the computational grid, ensuring that the resulting mesh accurately conforms to the problem geometry while capturing essential features such as boundaries, edges, and corners that influence the flow solution.

To facilitate the automatic discretisation of an arbitrary domain, the mathematical representation of the domain topology must be as general as possible. Such generality allows for the accurate representation of complex geometries with minimal manual intervention, enhancing adaptability across a wide range of applications – from simple geometries to highly intricate and irregular domains, such as aircraft. A generalised representation also ensures compatibility with various discretisation techniques and mesh generation algorithms, reducing the likelihood of errors during mesh generation.

In three dimensions, the domain to be discretised is considered as a region in $\mathbb{R}^3$ bounded by a general polyhedral structure, where the faces correspond to regions of curved surfaces intersecting along curves. The edges of this polyhedral representation lie on these intersection curves, forming the framework that defines the boundary of the computational domain (Faux and Pratt, 1980). In this context, the necessary portions of curves and surfaces that establish the domain boundaries are referred to as curve and surface components, respectively.

A surface component is defined as a region—or patch—on a surface, bounded by curve

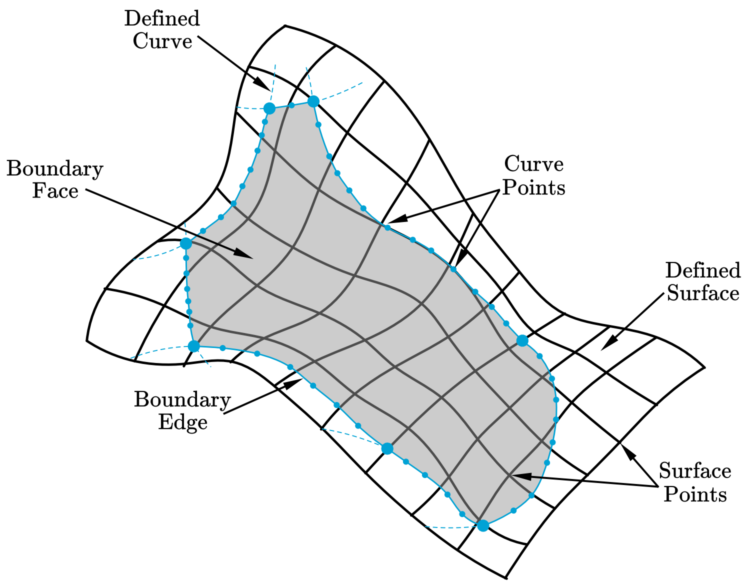


Figure 3.1: Example of an approximated surface component with corresponding curve components.

components. Each curve component is shared by two adjacent surface components and corresponds to a segment of the intersection curve between their respective supporting surfaces. The approximate representation of these boundary components is achieved using composite curves and surfaces, ensuring a consistent yet flexible discretisation that conforms to the underlying geometry.

An example illustrating the geometry of a surface component alongside its corresponding curve components is shown in Figure 3.1. This representation highlights that the boundaries of the composite surface are not necessarily constrained to align exactly with those of the original surface component. Instead, they serve as an adaptable framework that enables accurate mesh generation.

By defining the boundary structure in this manner, the meshing process ensures that critical flow regions—such as edges and corners—are properly resolved. This approach facilitates the generation of high-quality computational meshes that faithfully capture the geometric complexity of the domain while supporting efficient and accurate numerical simulations.

3.1.1 Curve Definition

Accurate curve definition is fundamental to ensuring that the computational mesh conforms precisely to the intended geometry. Curves define boundaries, edges, and intersections between surfaces, playing a crucial role in the quality and accuracy of the discretisation. A defined mathematical representation allows for the preservation of geometric fidelity whilst being efficient in numerical computations.

Curves are represented parametrically using a piecewise interpolation of cubic polynomials through an ordered set of data points, with their ordering determining the orientation of the curve. A widely used formulation is the Ferguson representation (Ferguson, 1964), which defines each curve segment using the position and tangent vectors at its endpoints,

$$\mathbf{r}(v) = \begin{bmatrix}1 & v & v^2 & v^3\end{bmatrix} \mathbf{C} \begin{bmatrix}\mathbf{r}^{(1)} \\ \mathbf{r}^{(2)} \\ \mathbf{t}^{(1)} \\ \mathbf{t}^{(2)}\end{bmatrix} \qquad 0 \le v \le 1. \tag{3.1}$$

Here, $\mathbf{r}$ represents the position in $\mathbb{R}^3$, while $\mathbf{r}^{(1)}$ and $\mathbf{r}^{(2)}$ denote the endpoints of the curve segment. The $\mathbf{t}^{(1)}$ and $\mathbf{t}^{(2)}$ vectors represent the tangent directions at these points with respect to v. The matrix $\mathbf{C}$, which defines the cubic interpolation, is given by

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix}. \tag{3.2}$$

The number and spatial distribution of data points are chosen to ensure that the interpolated curve accurately approximates the underlying surface intersections. This interpolation problem consists of fitting a parametric spline, defined piecewise, through a sequence of n points $\mathbf{r}_j$, where $j = 1, \ldots, n$, with $n \ge 3$.

To maintain slope continuity at interior points, the same tangent vector is applied to adjacent cubic segments. A simple procedure (Faux and Pratt, 1980) determines these tangent vectors to also enforce curvature continuity, resulting in a smooth interpolated curve. At the two endpoints, a zero-curvature condition is applied to close the system.

3.1.2 Surface Definition

Like curves, accurate surface representation is critical in computational geometry, ensuring that the discretised domain correctly captures the intended geometry. Surfaces define the boundaries of the computational domain, and their mathematical representation directly affects the quality of the generated mesh. A smooth and continuous surface definition is essential to prevent numerical errors and maintain proper element connectivity.

A composite surface is constructed by interpolating a series of quadrilateral patches through a structured set of data points $\mathbf{r}_{jk}$, where $j = 1, \ldots, m$ and $k = 1, \ldots, n$. First, two families of parametric curves are defined by interpolating cubic splines through the data points along constant j and k, respectively. The interpolation of these curves follows the procedure outlined in Section 3.1.1.

Each quadrilateral patch is then defined by its four bounding curves and the twist vectors at its corner points. The parametric representation of a surface patch is given by

$$r_i(v,w) = \begin{bmatrix} 1 & v & v^2 & v^3 \end{bmatrix} \mathbf{C} \begin{bmatrix} r_i^{(1)} & r_i^{(4)} & \frac{\partial r_i^{(1)}}{\partial w} & \frac{\partial r_i^{(4)}}{\partial w} \\ r_i^{(2)} & r_i^{(3)} & \frac{\partial r_i^{(2)}}{\partial w} & \frac{\partial r_i^{(3)}}{\partial w} \\ \frac{\partial r_i^{(1)}}{\partial v} & \frac{\partial r_i^{(4)}}{\partial v} & \frac{\partial^2 r_i^{(1)}}{\partial v \partial w} & \frac{\partial^2 r_i^{(4)}}{\partial v \partial w} \\ \frac{\partial r_i^{(2)}}{\partial v} & \frac{\partial r_i^{(3)}}{\partial v} & \frac{\partial^2 r_i^{(2)}}{\partial v \partial w} & \frac{\partial^2 r_i^{(3)}}{\partial v \partial w} \end{bmatrix} \mathbf{C}^T \begin{bmatrix} 1 \\ w \\ w^2 \\ w^3 \end{bmatrix} \qquad (3.3)$$

$$0 \le v, w \le 1,$$

where $r_i(v,w)$ represents the i-th component of the position vector $\mathbf{r}(v,w)$, with $i \in [1,3]$ corresponding to the Cartesian coordinates (x,y,z). The matrix $\mathbf{C}$ is the

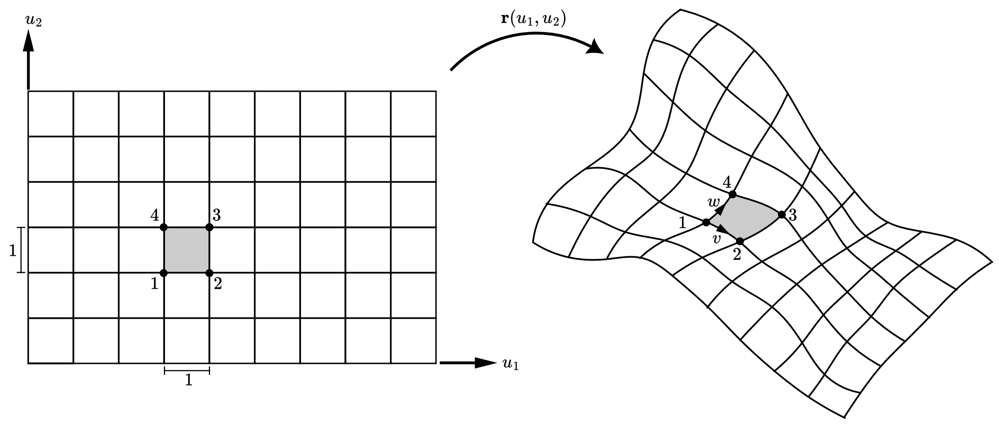


Figure 3.2: Mapping of a surface patch ($j = 4$, $k = 3$) from the structured parametric space (u_1, u_2) to the physical space using the transformation $\mathbf{r}(u_1, u_2)$. The highlighted quadrilateral patch in the parametric plane corresponds to a smoothly interpolated surface patch in physical space, ensuring continuity and geometric fidelity.

same cubic interpolation matrix defined in Equation (3.2). The four corner points of the patch are defined as

$$\mathbf{r}^{(1)} = \mathbf{r}(0,0), \quad \mathbf{r}^{(2)} = \mathbf{r}(1,0), \quad \mathbf{r}^{(3)} = \mathbf{r}(1,1), \quad \mathbf{r}^{(4)} = \mathbf{r}(0,1). \tag{3.4}$$

This formulation follows a Hermite interpolation between opposite boundaries of the patch (Coons, 1976). The twist vectors $\frac{\partial^2 r}{\partial v \partial w}$ at the corner points ensure second-order continuity across the surface, producing a smooth representation. These vectors are computed using an algorithm laid out by Coons (Coons, 1976).

To establish a global parametric representation on the surface plane, parametric coordinates (u_1, u_2) are introduced. For a given patch (j, k), these global coordinates relate to the local patch coordinates (v, w) by

$$u_1 = v + j - 1, \quad u_2 = w + k - 1, \tag{3.5}$$

as illustrated in Figure 3.2. This mapping defines a continuous function $\mathbf{r}(u_1, u_2)$, ensuring a structured parametric representation across the entire composite surface.

This parametric surface formulation provides a robust method for accurately defining curved boundaries in computational domains, ensuring that the generated mesh conforms to the underlying geometry.

3.2 Mesh Generation

Generating a high-quality discretisation of a computational domain with complex geometrical features is a critical step in CFD. Various mesh generation techniques have been developed to balance geometric accuracy, computational efficiency, and adaptability to flow features. The choice of method depends on the specific requirements of the problem, including the complexity of the geometry, numerical accuracy, and the need for local refinement.

One approach is octree meshing, which recursively subdivides the domain into hexahedral cells based on local geometric and solution-based criteria. This method efficiently captures complex geometries and allows for adaptive refinement, enabling finer resolution where necessary while maintaining coarser elements elsewhere. However, octree meshes introduce additional challenges, including complex data structures, solver compatibility issues, and difficulties in enforcing boundary conditions accurately.

One approach is octree meshing, which recursively subdivides the domain into hexahedral cells based on local geometric and solution-based criteria. This method efficiently captures complex geometries and allows for adaptive refinement, enabling finer resolution where necessary while maintaining coarser elements elsewhere. However, octree meshes introduce additional challenges, including the need for complex data structures to manage hierarchical cell relationships. More significantly, they can present solver compatibility issues, many CFD solvers are designed to work optimally with structured or unstructured meshes and may struggle to handle the hanging nodes and non-conforming interfaces commonly produced by octree methods. Additionally, the enforcement of boundary conditions can be less precise due to the Cartesian nature of octree cells, which may not align cleanly with curved or sloped domain boundaries.

In this work, unstructured mesh generation is employed due to its adaptability to complex geometries. Unlike structured and octree meshes, unstructured meshes can conform to arbitrary shapes without requiring the domain to be divided into predefined regions. This flexibility allows for localised refinement in areas with high flow gradients, such as boundary layers or shock waves, improving resolution where it is most needed without excessive computational overhead. Additionally, unstructured meshes enable rapid generation of high-quality grids for intricate geometries, making them a practical choice for complex CFD simulations.

The unstructured mesh employed in this thesis consists of linear tetrahedral elements, which are well-suited for complex geometries and adaptive refinement. These elements provide sufficient accuracy for CFD simulations – especially regarding shock capturing, while maintaining computational efficiency. Their flexibility, combined with the ability to refine locally without restructuring the entire mesh, makes tetrahedral meshes an ideal choice for the complex flow problems considered in this work.

3.2.1 Element Definition

An element within the computational mesh is defined by its size and orientation, both of which determine the local shape and spatial distribution of elements. The element sizes, denoted as δ_1, δ_2, and δ_3, specify the characteristic lengths along three mutually orthogonal directions. These directions, represented by the unit vectors $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$, form the principal axes of the local element frame and define the preferred stretching directions of the mesh. Together, these vectors constitute the columns of a rotation matrix $\mathbf{R}$, which aligns the local element frame with the global coordinate system.

The bounding box defined by δ_1, δ_2, and δ_3 encloses the element, with its edges aligned to the corresponding principal directions $\mathbf{e}_i$, as illustrated in Figure 3.3. When all three element sizes are equal, the mesh is isotropic, producing approximately equilateral tetrahedra in the vicinity of the point. In contrast, when the element sizes differ along each axis, the mesh exhibits anisotropic behaviour, allowing elements to stretch

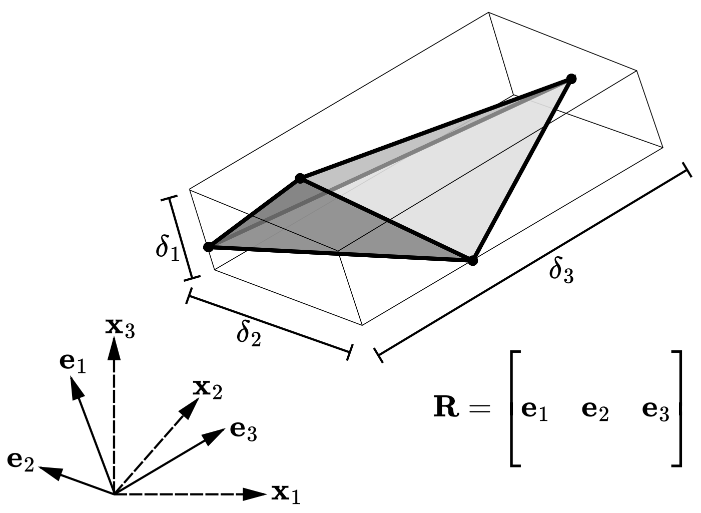


Figure 3.3: Illustration of local element definition, including element sizes and directional frame.

preferentially in specific directions. This directional refinement is particularly beneficial for capturing complex flow features, such as shock waves and boundary layers, where high gradients develop along preferred orientations.

3.2.2 Mesh parameters

To facilitate the mesh generation process, a set of mesh parameters is defined to describe the local geometric characteristics of the elements, including their shape, size, and orientation. A fundamental parameter in this process is the transformation matrix $\mathbf{T}$, which provides a mapping between the physical space and a normalised space where elements surrounding a point of interest are approximately equilateral with a unit average size. The transformation is constructed based on the principal directions $\boldsymbol{\alpha}_i$ and the corresponding element sizes δ_i. In three dimensions, $\mathbf{T}$ is represented by a symmetric $n \times n$ matrix, defined as the sum of the tensor product of the principal directions scaled by the inverse of the element sizes,

$$\mathbf{T} = \sum_{i=1}^{n} \frac{1}{\delta_i} \boldsymbol{\alpha}_i \otimes \boldsymbol{\alpha}_i. \tag{3.6}$$

This transformation effectively scales the physical domain such that the mesh elements in the normalised space are isotropic and of unit size. The transformation matrix $\mathbf{T}$ is not necessarily constant across the domain; instead, it varies spatially to accommodate changes in the required element sizes and orientations. The role of $\mathbf{T}$ is crucial in the mesh generation process, as the normalised space it defines provides a consistent frame in which elements can assessed as to whether they are correctly sized and shaped.

In addition to the transformation matrix, the local element characteristics can also be described using a metric tensor, $\boldsymbol{\mathcal{M}}$, which encodes both the element shape and size at a given point in the domain. The metric tensor defines a mapping from the standard Euclidean space to a local Riemannian space, where the element size and orientation are explicitly prescribed.

The metric tensor $\boldsymbol{\mathcal{M}}$ is a symmetric, positive-definite tensor, ensuring that all prescribed element sizes remain strictly positive and that the local distance metric is well-defined. A key advantage of working in a metric space is that fundamental operations such as metric interpolation and metric intersection can be performed in a well-posed manner. The metric tensor is given by

$$\boldsymbol{\mathcal{M}} = \mathbf{R}\boldsymbol{\Lambda}^{-2}\mathbf{R}^{T} \tag{3.7}$$

where $\boldsymbol{\Lambda}^{-2} = \mathrm{diag}\,(1/\delta_1^2, 1/\delta_2^2, 1/\delta_3^2)$. The element sizes δ_i are always arranged in ascending order to ensure a consistent definition of the principal directions. Consequently, the basis vectors forming $\mathbf{R}$ are also ordered accordingly, maintaining alignment with the sorted spacings.

To recover the element sizes and directions from a given metric tensor, the eigenvalues and eigenvectors can be computed. The eigenvectors correspond to the principal directions, while the eigenvalues yield the squared inverse of the element sizes, δ_i^{-2}.

3.2.2.1 Metric Interpolation

In the context of anisotropic mesh generation, element size and orientation are typically defined only at discrete points in the domain. To determine the desired element characteristics at an arbitrary location, an appropriate interpolation scheme is required. This is achieved through metric interpolation, which ensures a smooth and gradual variation of the metric tensor across the domain, and subsequently a smooth and gradual variation in the elements.

The objective of metric interpolation is to construct a continuously varying metric tensor $\boldsymbol{\mathcal{M}}(t)$ along a segment connecting two points, P_1 and P_2, each associated with its own metric tensor, $\boldsymbol{\mathcal{M}}_1$ and $\boldsymbol{\mathcal{M}}_2$, respectively. The interpolated metric must satisfy the boundary conditions $\boldsymbol{\mathcal{M}}(0) = \boldsymbol{\mathcal{M}}_1$ and $\boldsymbol{\mathcal{M}}(1) = \boldsymbol{\mathcal{M}}_2$ while varying smoothly along the segment to prevent abrupt changes that could degrade element quality.

For isotropic metrics, where $\boldsymbol{\mathcal{M}}_1$ and $\boldsymbol{\mathcal{M}}_2$ are scalar multiples of the identity matrix, interpolation reduces to a simple linear interpolation of the element spacing. However, in the general anisotropic case, both size and directional variations must be considered. The interpolated metric along the segment is given by

$$\boldsymbol{\mathcal{M}}(t) = \left((1-t)\,\boldsymbol{\mathcal{M}}_1^{-\frac{1}{2}} + t\,\boldsymbol{\mathcal{M}}_2^{-\frac{1}{2}}\right)^{-2}, \tag{3.8}$$

where $\boldsymbol{\mathcal{M}}_1$ and $\boldsymbol{\mathcal{M}}_2$ are the metrics at the endpoints, and $t \in [0,1]$ is the interpolation weighting. This formulation ensures smooth variation in both size and orientation across the segment, avoiding discontinuities. Figure 3.4 illustrates this process, showing a gradual transition between two endpoint metrics.

This formulation extends naturally to higher-dimensional domains by performing a sequence of one-dimensional interpolations. In three dimensions, the interpolation of a metric in a generic point $\boldsymbol{x}$ contained in an element of the background mesh with nodes $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{x}_3$ and $\boldsymbol{x}_4$ is performed as follows. First, the intersection of the line connecting $\boldsymbol{x}_4$ and $\boldsymbol{x}$ with the triangular face formed by nodes $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ and $\boldsymbol{x}_3$, denoted as $\boldsymbol{x}_{123}$,

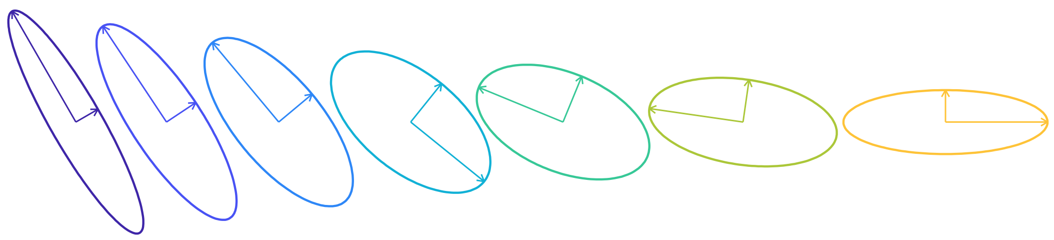


Figure 3.4: Illustration of metric interpolation along a segment. The blue and yellow ellipses represent the metrics at the endpoints, with the interpolated metrics smoothly transitioning between them.

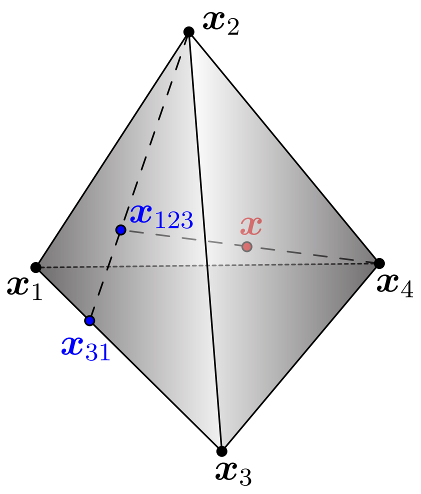


Figure 3.5: Schematic representation of the metric interpolation for a point inside a tetrahedral element.

is computed. Next, the intersection of the line connecting $\boldsymbol{x}_2$ and $\boldsymbol{x}_{123}$ with the edge connecting nodes $\boldsymbol{x}_3$ and $\boldsymbol{x}_1$, denoted as $\boldsymbol{x}_{31}$, is computed. The metric interpolation is applied to the metrics at nodes $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$ to compute the metric at $\boldsymbol{x}_{31}$. Then the metric interpolation is applied to the metrics at nodes $\boldsymbol{x}_2$ and $\boldsymbol{x}_{31}$ to compute the metric at $\boldsymbol{x}_{123}$. Finally, the metric interpolation is applied to the metrics at nodes $\boldsymbol{x}_4$ and $\boldsymbol{x}_{123}$ to compute the desired metric at $\boldsymbol{x}$. The process is illustrated in Figure 3.5.

It should be noted, however, that this interpolation scheme is non-associative. When interpolating between three or more metrics, the final result depends on the order in which pairwise interpolations are performed. For example, interpolating first between $\mathcal{M}_1$ and $\mathcal{M}_2$, and then between the result and $\mathcal{M}_3$, may not yield the same result as interpolating first between $\mathcal{M}_2$ and $\mathcal{M}_3$, followed by interpolation with $\mathcal{M}_1$. This non-associativity arises from the non-linear nature of the interpolation formula,

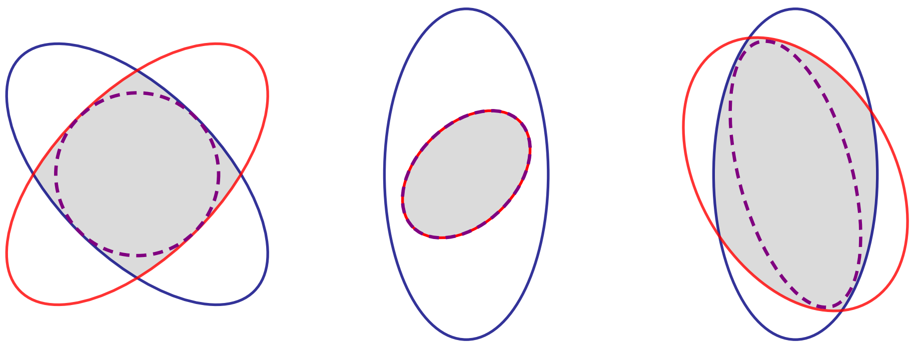


Figure 3.6: Illustration of metric intersection, where the red and blue ellipsoids represent the geometric interpretations of $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively. The dashed purple ellipsoid represents the resulting metric $(\mathcal{M}_1 \cap \mathcal{M}_2)$, which corresponds to the largest ellipsoid contained within their intersection.

which operates in the space of square-root metrics rather than directly on the metric tensors. Despite this limitation, this approach remains widely used due to its ability to produce smooth, well-graded metric fields that preserve anisotropic characteristics while maintaining computational efficiency.

3.2.2.2 Metric Intersection

In anisotropic mesh generation, different mesh size and orientation constraints may exist at the same location due to multiple geometric or solution-based requirements. When this occurs, a conservative approach is required to ensure that the final element sizes satisfy all imposed constraints. This is achieved by computing the intersection of the corresponding metric tensors, determining the largest ellipsoid that fits entirely within the intersection of the individual ellipsoids associated with each metric.

Each metric tensor can be visualised as an ellipsoid in three-dimensional space, where the axes and lengths of the ellipsoid represent the preferred directions and element sizes prescribed by the metric. The goal of metric intersection is to determine the largest ellipsoid that remains fully contained within the overlapping region of the ellipsoids associated with $\mathcal{M}_1$ and $\mathcal{M}_2$, ensuring that the most restrictive directional spacing constraints are satisfied. This process is illustrated in Figure 3.6.

Given two metric tensors, $\mathcal{M}_1$ and $\mathcal{M}_2$, defined at the same location, the intersected metric $\boldsymbol{\mathcal{M}}_1 \cap \boldsymbol{\mathcal{M}}_2$ is determined by ensuring that no directional spacing exceeds the smallest prescribed value from either metric.

To compute the intersection, the first step is to express both metric tensors in a common basis. This is achieved by finding a basis in which both $\boldsymbol{\mathcal{M}}_1$ and $\boldsymbol{\mathcal{M}}_2$ are diagonal. The transformation is obtained by computing the matrix product

$$\boldsymbol{\mathcal{N}} = \boldsymbol{\mathcal{M}}_1^{-1} \boldsymbol{\mathcal{M}}_2, \tag{3.9}$$

where the real eigenvectors of $\boldsymbol{\mathcal{N}}$ define the shared basis. Let $\mathbf{P}$ be the matrix whose columns are these eigenvectors, denoted $\mathbf{e}_i$. In this common basis, the metric tensors take the form

$$\boldsymbol{\mathcal{M}}_1 = \mathbf{P}^{-T} \boldsymbol{\lambda} \mathbf{P}^{-1} \quad \text{and} \quad \boldsymbol{\mathcal{M}}_2 = \mathbf{P}^{-T} \boldsymbol{\mu} \mathbf{P}^{-1}, \tag{3.10}$$

where $\boldsymbol{\lambda} = \operatorname{diag}(\lambda_1, \lambda_2, \lambda_3)$ and $\boldsymbol{\mu} = \operatorname{diag}(\mu_1, \mu_2, \mu_3)$ contain the spectral values of $\boldsymbol{\mathcal{M}}_1$ and $\boldsymbol{\mathcal{M}}_2$, respectively. These values are computed as

$$\lambda_i = \mathbf{e}_i^T \boldsymbol{\mathcal{M}}_1 \mathbf{e}_i \quad \text{and} \quad \mu_i = \mathbf{e}_i^T \boldsymbol{\mathcal{M}}_2 \mathbf{e}_i, \quad \text{for} \quad i = 1, \ldots, 3. \tag{3.11}$$

The intersected metric $\boldsymbol{\mathcal{M}}_1 \cap \boldsymbol{\mathcal{M}}_2$ is then computed by selecting, for each principal direction, the most restrictive spacing requirement – corresponding to the larger of the two spectral values. In the shared eigenbasis, the intersection metric takes the form

$$\boldsymbol{\mathcal{M}}_1 \cap \boldsymbol{\mathcal{M}}_2 = \mathbf{P}^{-T} \, \boldsymbol{\Lambda}_{\max} \, \mathbf{P}^{-1}, \tag{3.12}$$

where $\boldsymbol{\Lambda}_{\max} = \operatorname{diag}\left(\max\left\{\lambda_1, \mu_1\right\}, \max\left\{\lambda_2, \mu_2\right\}, \max\left\{\lambda_3, \mu_3\right\}\right)$. This formulation ensures that in each principal direction, the final element size is dictated by the smallest prescribed spacing, enforcing the most restrictive constraints. The resulting metric represents the largest ellipsoid that remains entirely within the geometric intersection of the ellipsoids associated with $\boldsymbol{\mathcal{M}}_1$ and $\boldsymbol{\mathcal{M}}_2$, guaranteeing that the final mesh spacing satisfies both constraints.

It is important to note since metric intersection is non-associative – the result can depend on the order in which metrics are intersected. When more than two metrics need to be combined, the process no longer guarantees optimality, as the resulting metric depends on the sequence of pairwise intersections.

Although metric intersection is non-associative, the variation in the resulting metric due to different intersection orders is typically small in practical applications. This is because the principal directions of the intersected metrics are generally well-aligned when derived from similar geometric or solution-driven constraints. While the precise metric obtained may differ depending on the sequence of intersections, these variations are often insignificant from a mesh generation perspective, as they result in element anisotropies that are sufficiently close to one another. Consequently, despite the theoretical dependence on the intersection order, the impact on the final mesh quality is usually negligible.

An example of this non-associative behaviour is illustrated in Figure 3.7, where different intersection sequences produce slightly different resultant metrics. While minor discrepancies exist, the resulting element distributions remain functionally close for meshing purposes, underscoring the robustness of the intersection process despite its order dependence.

3.2.3 Mesh Control

Effective mesh control is essential for ensuring that the generated mesh meets the accuracy and efficiency requirements for a given simulation. Proper control over element size and orientation allows critical flow features, such as shock waves, to be captured with sufficient resolution while avoiding excessive computational costs. This is achieved by defining a spatial distribution of mesh parameters that governs element refinement across the domain.

Mesh control is implemented using two primary methods: the background mesh and

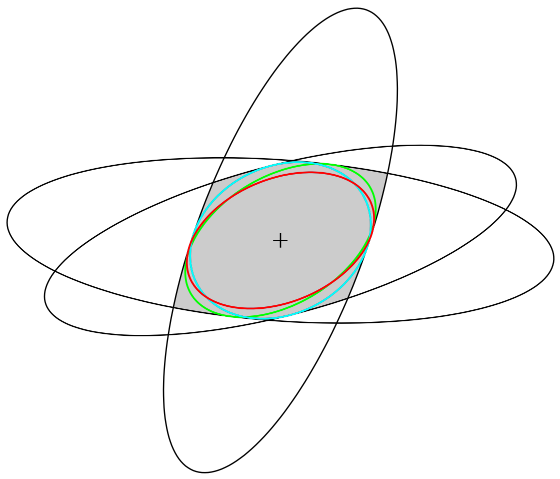


Figure 3.7: Illustration of the non-associativity of metric intersection. The black ellipses represent the original metrics, while the smaller ellipses show the resulting metric when different intersection orders are applied. The final metric depends on the sequence of operations, demonstrating that pairwise intersections do not necessarily yield an optimal result.

sources. The background mesh provides a spatial spacing field for defining the desired element sizing at discrete points of the domain, while sources allow for localised refinement in specific regions of interest. These methods can be used independently or in combination to achieve optimal mesh refinement and distribution.

3.2.3.1 Sources

For complex geometries, refining the mesh in key regions is necessary to ensure accurate numerical solutions and efficient computation. Sources provide a simple and effective means of enforcing local mesh refinement, offering precise control over element sizes in critical areas such as leading and trailing edges of aerodynamic surfaces. This method is particularly user-friendly compared to alternative approaches, as it allows mesh parameters to be specified intuitively based on geometric proximity.

While sources are primarily used to define isotropic refinement, they can also introduce limited anisotropic refinement. For instance, elements can be elongated along the direction of a line source to better capture flow-aligned features. However, this capability is much more restricted compared to the background mesh, which provides full control over element anisotropy. In this work, sources are only considered in the context of prescribing isotropic spacing, and so only a single spacing value, δ, is considered. The element spacing $\delta(d)$ at a distance d from a source is given by

$$\delta(d) = \begin{cases} \delta_0 & \text{if } d \leq r, \\ \delta_0 e^{\ln(2)\frac{d-r}{R-r}} & \text{otherwise,} \end{cases} \tag{3.13}$$

where δ_0, r, and R are user-defined source parameters. Within a radius r, the element spacing remains constant at δ_0. Beyond this region, spacing increases exponentially, with R defining the distance at which the element size has doubled to $2\delta_0$. The element size continues to increase following this exponential growth until it reaches the global maximum spacing δ_{max}.

Since sources define localised spacing fields that radiate outward, multiple sources may influence the desired element size at a given point in the domain. In such cases, the final element spacing is determined by taking the minimum value prescribed by all contributing sources. This ensures that the most restrictive refinement requirement is enforced, allowing fine-scale features to be captured while maintaining smooth transitions between different refinement regions. The influence of multiple sources is illustrated in Figure 3.8, which demonstrates the interaction of overlapping source regions and the resulting mesh refinement.

This source-based approach naturally extends to line and surface-based sources. For these cases, the parameters δ_0, r, and R are assigned at each node of the source geometry. To determine the element size at an arbitrary point, p, in the domain, the point is projected onto the source geometry. The three source parameters at the projected point are then linearly interpolated from the user-defined source nodes. The spacing $\delta(d)$ is then evaluated based on the distance between p and its projected location, using the interpolated source parameters.

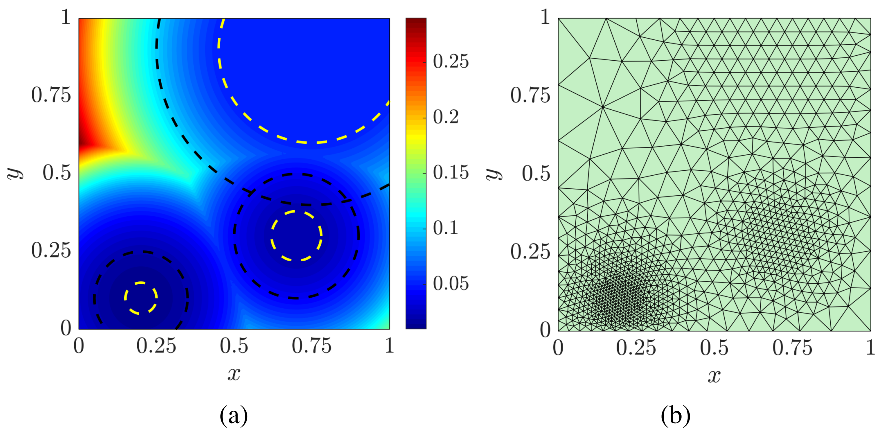


Figure 3.8: (a) Illustration of three point sources defining the local element spacing δ, with the colour map representing the prescribed spacing value. The dashed circles indicate the two radii of each source, with the final spacing at any given point determined by the minimum prescribed value. (b) The resulting mesh generated using the spacing field from (a).

3.2.3.2 Background Mesh

The background mesh provides a structured mesh for defining the size and orientation of the mesh element in the computational domain. Unlike source-based methods for controlling the mesh, which have limited anisotropic mesh control, the background mesh allows for a full specification of mesh anisotropy, making it particularly useful for generating meshes that accurately resolve sharp flow features such as shocks, where directional stretching of mesh elements can improve numerical efficiency.

At each node of the background mesh, the mesh parameters α_i and δ_i are specified, controlling mesh element orientation and size, respectively. To determine the desired mesh element characteristics at an arbitrary point p, the background mesh element containing p is first identified. If isotropic mesh refinement is used, a simple linear interpolation is performed using the mesh spacing defined at the nodes of the enclosing background mesh element.

For anisotropic mesh refinement, the mesh metric tensor must be computed at each node of the background mesh element using Equation (3.7). The mesh metric at point p is then obtained through mesh metric interpolation, as described in Section 3.2.2.1.

The mesh element characteristics can then be computed by extracting the eigenvalues and eigenvectors of the metric tensor at p.

3.2.3.3 Curvature Control

During both curve and surface mesh discretisation, it is essential to ensure that the mesh accurately represents the underlying geometry. A primary challenge in meshing curved geometries lies in maintaining sufficient resolution in regions of high curvature. If the element spacing is too large relative to the local curvature, the discretised representation may fail to capture geometric features accurately, leading to significant approximation errors. These inaccuracies can degrade the fidelity of numerical simulations, particularly in applications where boundary precision is crucial, such as CFD. To mitigate this issue, curvature control is applied to enforce appropriate local element spacing.

The curvature control procedure begins with an analysis of local curvature at discrete sampling points along a curve or within the parametric plane of a surface. The local radius of curvature, R, is computed at each sampled point. For curves, this radius is obtained directly from the derivatives of the curve representation. For surfaces, curvature varies with direction, and so, the minimum radius is selected, as it imposes the most restrictive element sizing requirement.

Two curvature-based constraints regulate element spacing. The first constraint limits the maximum sector angle, where a user-defined maximum sector angle, $\theta_{\max}$, governs the angular resolution of the discretisation. Given the local radius of curvature R, the largest permissible chord length – and subsequent spacing, within the circle of curvature is constrained by

$$\delta_{\max} \leq 2R \sin\left(\frac{\theta_{\max}}{2}\right). \tag{3.14}$$

This ensures that each mesh element subtends an angle within the prescribed limit, preventing excessively large elements in high-curvature regions. The second constraint limits the maximum perpendicular deviation h between the true geometry and the discretised representation, ensuring that the deviation remains within a user-specified threshold to prevent excessive geometric distortion. This results in the constraint

$$\delta_{\max} \leq \sqrt{8Rh}. \tag{3.15}$$

3.2.4 Curve Discretisation

The first stage of the discretisation procedure involves dividing the boundary curve components into smaller segments by placing nodes along each curve according to the locally prescribed mesh spacing. These nodes are then connected by straight lines to form the discretised representation of the curve.

The process begins with the recursive subdivision of each cubic curve segment until its length is smaller than the globally defined minimum spacing. The length of each cubic segment is determined numerically and the positions and tangent vectors at newly generated points are directly obtained by interpolating the original curve definition using Equation 3.1.

For every point along the curve, indexed by $j = 1, 2, \ldots, n$ – including both the original curve-defining nodes and those generated to meet the above length criterion, the transformation tensor $\mathbf{T}_j$ is determined based on the specified mesh parameters. This transformation is then applied to both the position and tangent vectors, yielding $\hat{\mathbf{r}}_j = \mathbf{T}_j \mathbf{r}_j$ and $\hat{\mathbf{t}}_j = \mathbf{T}_j \mathbf{t}_j$. The transformed vectors define a new spline curve within the normalised space. However, due to the approximate nature of this transformation, the new curve generally exhibits curvature discontinuities, even if the curvature of the original curve varies smoothly.

To finalise the discretisation, the length of the curve in the normalised space is computed and subdivided into approximately unit-length segments. For each newly generated node, the corresponding cubic segment is identified, and its parametric coordinate is determined. This allows for an accurate mapping of the new nodes back to physical space, ensuring that the discretised curve adheres to the prescribed mesh spacing while accurately representing the underlying geometry.

The final set of nodes, now appropriately spaced according to the mesh specification, is connected by straight lines to form the fully discretised curve.

3.2.5 Surface Discretisation

The second stage of mesh generation involves the discretisation of the surface, ensuring that the generated elements conform to the prescribed mesh spacing while accurately capturing the underlying geometry. The surface mesh generation process is carried out within three distinct spaces, each serving a specific role in the procedure.

The first space is the parametric plane, where the surface is denoted by S^*. In this space, the surface mesh exists as a two-dimensional triangular mesh, where the parametric coordinates of the nodes define the surface mesh, enabling a straightforward construction of a three-dimensional surface mesh that conforms to the underlying boundary geometry. This parametric mesh is then mapped to the second space, the physical space, using the transformation $\mathbf{r}(u_1, u_2)$, as detailed in Section 3.1.2. This transformation ensures that a well-defined triangular mesh in the parametric plane is mapped onto a valid surface triangulation in physical space. The resulting surface in physical space is denoted by S and represents the actual geometry within the computational domain.

A third space, known as the normalised space, is used to enforce the prescribed element sizes and anisotropic spacing requirements. The surface in this space, denoted by $\hat{S}$, is obtained through the transformation $\mathbf{T}(\mathbf{x})$, which maps the physical space into the normalised space where elements are approximately equilateral and of unit size. The transformation from the parametric plane to the normalised space is described by the composite function

$$\boldsymbol{\mathcal{T}}(u_1, u_2) = \mathbf{T} \circ \mathbf{r}(u_1, u_2). \tag{3.16}$$

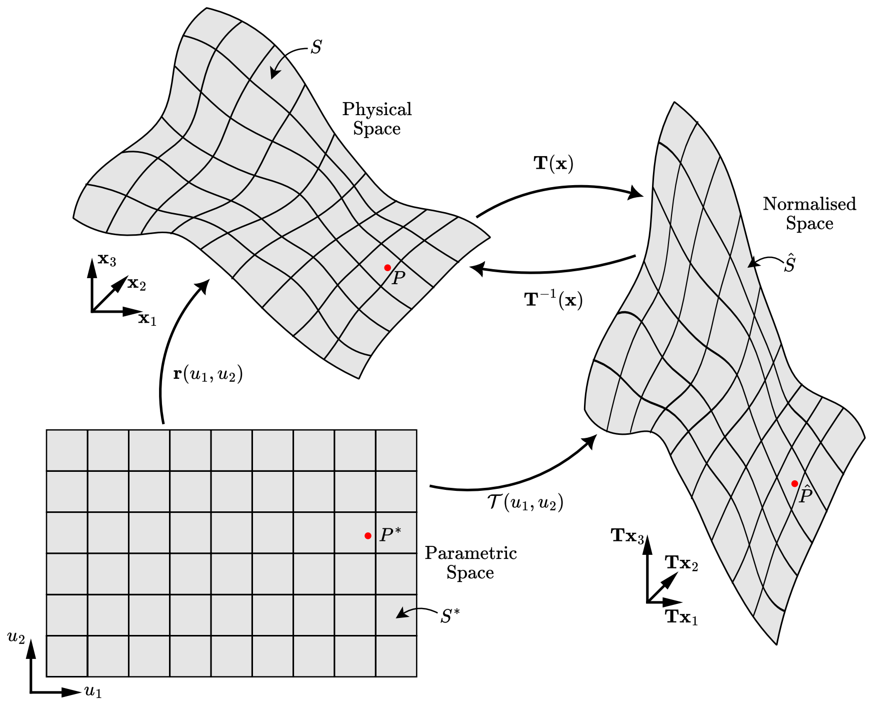


Figure 3.9: Mapping between the parametric plane, physical space, and normalised space in the surface discretisation process.

This function maps coordinates from the parametric plane to the normalised space directly, ensuring that spacing constraints are satisfied in the normalised space.

By describing the meshed nodes in the parametric plane and subsequently transforming them into the normalised space, the surface discretisation process aims to ensure that when the mesh is mapped into physical space, the elements maintain the correct anisotropic spacing and accurately represent the true surface geometry. The relationship between the parametric plane, physical space, and normalised space is illustrated in Figure 3.9.

3.2.5.1 Computation of Parametric Mesh Parameters

To construct the triangular mesh in the parametric plane, it is necessary to determine an appropriate spatial distribution of the mesh parameters. Similarly to the mesh parameters in the 3-dimensional volume, these parameters consist of two mutually orthogonal directions, denoted α_i^* (where $i = 1, 2$), and the corresponding element sizes δ_i^*. The two-dimensional mesh parameters in the parametric plane are derived from the distribution of the three-dimensional mesh parameters while accounting for the distortion introduced by the surface mapping.

Consider a curve in the parametric plane passing through P^* with a unit tangent vector $\boldsymbol{\beta} = (\beta_1, \beta_2)$. Under the transformation $\mathcal{T}(u_1^P, u_2^P)$, this curve is mapped into the normalised space, passing through the corresponding point $\mathcal{T}_P P$. The arc lengths in the parametric and normalised spaces, denoted as ds^* and $d\hat{s}$, respectively, are related by the expression

$$(d\hat{s})^2 = \left\{ \sum_{i=1}^{2} \sum_{j=1}^{2} \frac{\partial \mathcal{T}}{\partial u^i} \frac{\partial \mathcal{T}}{\partial u^j} \beta^i \beta^j \right\} (ds^*)^2. \tag{3.17}$$

Assuming that this relation also applies to the mesh spacings, the spacing δ_β^* along a given direction $\boldsymbol{\beta}$ in the parametric plane can be computed as

$$\frac{1}{{\delta_\beta^*}^2} = \sum_{i=1}^{2} \sum_{j=1}^{2} \frac{\partial \mathcal{T}}{\partial u^i} \frac{\partial \mathcal{T}}{\partial u^j} \beta^i \beta^j. \tag{3.18}$$

To determine the principal directions and corresponding element sizes in the parametric plane, the values of δ_β^* are analysed to identify their extrema. This requires computing the eigenvalues and eigenvectors of the symmetric 2×2 matrix formed from the transformation gradients. The eigenvectors provide the principal directions α_i^*, while the eigenvalues define the corresponding element sizes δ_i^*.

3.2.5.2 Advancing Front Method

The advancing front method is a widely used approach for unstructured mesh generation, particularly suited for complex geometries. It constructs the mesh by progressively advancing from the boundaries of the computational domain toward the interior. At any stage of the process, the domain is divided into two regions: the meshed region, where triangulation is complete, and the unmeshed region, where element generation is ongoing. This localised approach ensures efficient memory usage while producing high-quality elements with well-shaped triangles or tetrahedra.

The process begins by defining the initial front, which consists of the boundary nodes obtained from the curve discretisation discussed in Section 3.2.4. These nodes define the outer boundary of the unmeshed region and serve as the starting point for element generation. However, as the boundary nodes are initially defined in physical space, their corresponding coordinates in the parametric space must first be determined. Since the mapping $\mathbf{r}(u_1, u_2)$ between the parametric and physical spaces is generally non-invertible in closed form, the parametric coordinates of these nodes are computed numerically through an iterative procedure. Once determined, the nodes are connected by straight-line segments, forming the initial active front.

Each segment of the front is considered *active*, indicating its availability for element generation. As the meshing process advances, active segments are continuously updated, and newly created elements remove existing segments from the front while introducing new ones. This dynamic updating continues until the entire domain is meshed and no active segments remain.

With the initial front established, the advancing front method proceeds by selecting an active segment as the base of a new triangular element. The segment chosen is the shortest active segment in the parametric space, denoted AB^*, as selecting the shortest segment promotes element quality, particularly in regions where spacing transitions occur.

Once the base segment is selected, the transformation matrix $\mathcal{T}$ must be determined to ensure the new element conforms to the prescribed mesh spacing. However, since the transformation varies continuously across the domain, there is no exact, universally correct mapping. Ideally, this transformation is evaluated at the midpoint of the segment in physical space, denoted M, as it provides a representative measure of the desired element anisotropy. However, the parametric coordinates of M are unknown due to the non-invertibility of the surface mapping function, requiring an iterative approach to approximate its location.

To approximate them, the midpoint of the segment in parametric space, M^*, is first identified and mapped to physical space using $\mathbf{r}(M^*)$. In general, this mapped point will not divide the physical segment evenly. A tolerance is prescribed to ensure an acceptable placement, and if the condition is not met, the parametric coordinates of M^* are iteratively adjusted along the segment until a suitable midpoint is found. Once this condition is met, the mapping that defines the normalised space for this element creation, $\boldsymbol{\mathcal{T}}(M^*)$, is finalised, allowing the advancing front procedure to generate the new element.

With the base segment AB^* selected and the transformation matrix $\boldsymbol{\mathcal{T}}(M^*)$ established, the next step is to determine the optimal position for the third node, denoted P^*, to form a new triangular element.

To begin, an equilateral triangle is first constructed in the parametric plane, ensuring that the new node P^* is positioned at a distance $|AB^*|$ from both A^* and B^* into the active region. This initial placement defines an approximate location for the third node before mapping it to the normalised space. Once the parametric coordinates of P^* are established, the mapping $\boldsymbol{\mathcal{T}}(M^*)$ is applied to obtain the corresponding point $\hat{P}$ in the normalised space.

The segment length in the normalised space, $\hat{AB}$, is then computed, allowing the determination of the desired position for $\hat{P}$. Ideally, $\hat{P}$ should lie on the line perpendicular to the segment $(\hat{AB})$, passing through its midpoint, and at a distance δ_1 from both $\hat{A}$ and $\hat{B}$. The value of δ_1 is empirically determined based on the segment length $L = |\hat{AB}|$, where

$$\delta_1 = \begin{cases} \sqrt{2}L, & \text{if } L < 1/\sqrt{2}, \\ 1, & \text{if } 1/\sqrt{2} \leq L \leq \sqrt{2}, \\ L/\sqrt{2}, & \text{if } L > \sqrt{2}. \end{cases} \tag{3.19}$$

However, due to the effects of the mapping, $\hat{P}$ is often not exactly at this desired location. To correct this, a Newton-Raphson iterative procedure is employed to refine the parametric coordinates of P^*, ensuring that its mapped location aligns with the desired position in normalised space. The iteration continues until the error falls within a specified tolerance.

With the ideal location of the third vertex in the parametric plane now determined, the next step is to evaluate the best choice for the actual third vertex of the new element. While P^* represents the optimal placement in isolation, the final selection must account for the existing mesh structure. Potential candidates include active nodes (nodes that are part of the active front) in the vicinity of P^*. To determine these candidate nodes, an ellipse is constructed around P^* in the parametric plane. The radii and directions of this ellipse are defined by the local mesh parameters at P^*, as given by Equation 3.18. Any active nodes located within this elliptical region are identified and labelled as Q_i. A list is then created containing all candidate points. The active nodes Q_i are ordered such that the furthest point from P^* appears at the head of the list. The ideal point P^* is then placed after the Q_i points. The best candidate for the third vertex is selected as the first point in the ordered list that forms a valid triangular element. An element is considered valid if none of its newly created edges intersect with any existing segments in the active front.

Once a valid third vertex is chosen, the new triangle is stored, and the front is updated accordingly by adding new active segments and removing those that are no longer needed. This process is repeated iteratively until no active region remains, at which point the entire planar domain – and consequently the physical surface – is fully meshed.

3.2.6 Volume Discretisation

With the surface discretisation complete, the next stage is to generate the volume mesh by filling the interior of the domain with elements. This process must ensure that the internal discretisation conforms to the prescribed mesh parameters while maintaining consistency with the surface mesh.

Two common approaches for volume mesh generation are advancing-front methods and Delaunay triangulation with point insertion. Similarly to how advancing front worked for the surface mesh generation discussed in Section 3.2.5.2, the technique grows the mesh into the domain from the boundary surfaces. The method is inherently local, operating on only a small region of the domain at a time. This approach makes the method highly memory-efficient and consistently produces meshes of high quality.

In this work, however, the volume mesh is generated entirely using a Delaunay-based approach with point insertion. This method incrementally constructs the mesh by inserting points and forming tetrahedral elements, while ensuring the Delaunay criterion is satisfied throughout the process. By dynamically adapting to complex geometries and maintaining an optimal element distribution, this approach provides a robust and efficient method for unstructured volume mesh generation.

3.2.6.1 Voronoi Diagrams and Delaunay Triangulation

The foundation of the Delaunay scheme lies in the construction of Voronoi diagrams. Given an initial set of points $\{P_K \in \mathbb{R}^d\}$, $K \in [1, N]$, as shown in Figure 3.10(a), each point P_K is associated with a region V_K called its Voronoi region. The Voronoi region of a point consists of all locations in the domain that are closer to P_K than to any other point, forming a set of convex, non-overlapping polygons in 2D (or polyhedra in 3D), as illustrated in Figure 3.10(b). The collection of all Voronoi regions is termed the Voronoi diagram.

The Delaunay triangulation is then defined as the dual of the Voronoi diagram: two points are connected by an edge if and only if their Voronoi regions share a common face, forming a set of non-overlapping triangles or tetrahedra, as depicted in Figure 3.10(c). This triangulation is unique for a given point distribution (provided four of the points do not lie on the circumference of the same circle) and has two desirable properties, including the in-circle criterion, which states that the circumcircle (or circumsphere in 3D) of any element in a Delaunay mesh contains no other mesh points – seen in

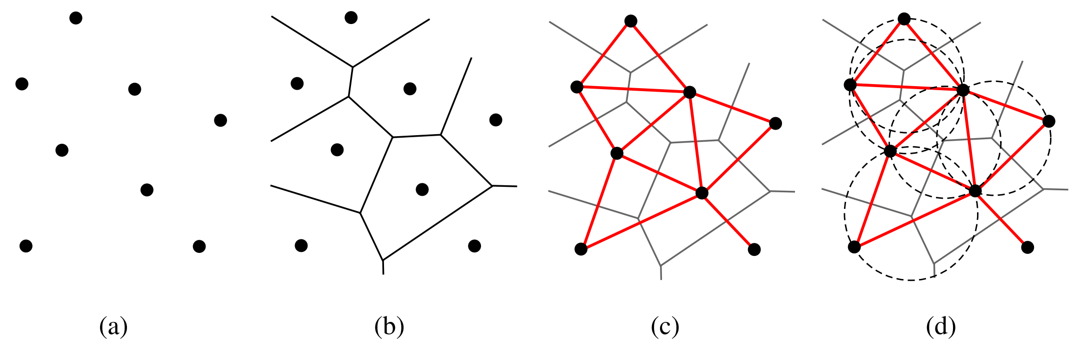


Figure 3.10: Illustration of (a) an initial set of points, (b) the corresponding Voronoi diagram, and (c) the resulting Delaunay triangulation, where edges connect points with a common Voronoi face. (d) Demonstration of the in-circle criterion, which ensures that no other points lie inside the circumcircles of any Delaunay triangles.

Figure 3.10(d).

In planar triangulations, Delaunay meshes maximise the minimum internal angle among all possible triangulations, thereby mitigating the formation of excessively acute or narrow triangles (Shewchuk, 2002; Lawson, 1977; Lee and Lin, 1986). This max–min angle property, however, does not extend naturally to 3D volumetric tetrahedralisations, whether in terms of planar angles or dihedral angles (Cheng et al., 2009). Despite this, Delaunay meshes in three dimensions remain widely used due to their robustness and algorithmic efficiency. However, they do not inherently control element quality in 3D, and mesh enhancement techniques are often applied in practice to ensure and improve the quality of the elements – Section 3.2.7.

3.2.6.2 Delaunay Triangulation Implementation

The Delaunay mesh generation process consists of two key stages. First, Boundary Triangulation, where the initial Delaunay triangulation is constructed using only the boundary points obtained from the surface discretisation. Then, point creation, where interior points are incrementally generated and subsequently inserted into the mesh to refine element sizes whilst maintaining the Delaunay criterion.

As outlined earlier, the Delaunay triangulation method incrementally constructs the mesh by inserting points and forming tetrahedral elements while ensuring the Delaunay criterion is satisfied at each step. Point insertion is the core operation of Delaunay-based mesh generation, allowing the mesh to be dynamically refined to meet the prescribed element sizes. When a new point x_i is inserted, the mesh must be updated while maintaining the Delaunay criterion. The procedure follows these steps, as illustrated in Figure 3.11.

1. **Identify Affected Elements:** The first step is to locate all elements whose circumcircle (or circumsphere in 3D) contains the new point x_i. These elements violate the Delaunay criterion and must be removed.

2. **Remove Invalid Elements:** The affected elements are deleted, creating a cavity around x_i. The boundary of this cavity consists of the edges (or faces) that were shared by the removed elements but remain in the mesh.

3. **Form New Elements:** New elements are created by connecting x_i to each edge (or face) on the cavity boundary, restoring connectivity while ensuring that no new element violates the Delaunay criterion.

4. **Update the Mesh:** After inserting a point, some edges (or faces) may still violate the Delaunay criterion. Edge flipping (or face swapping in 3D) is performed to restore a valid Delaunay triangulation.

The first step in volume mesh generation is to construct an initial triangulation that spans the entire computational domain. This is achieved by first creating a tessellated convex hull that encloses the domain, as shown in Figure 3.12(a). Then, each node of the surface mesh points is inserted into the domain one at a time. The insertion follows the standard point insertion procedure (described above), ensuring that the triangulation is valid – seen in Figures 3.12(b) and 3.12(c). The process continues until all boundary points are inserted and tessellated. However, this initial tessellation may not fully conform to the surface mesh. To correct this, boundary recovery techniques are applied to enforce conformity between the surface mesh and the internal volume

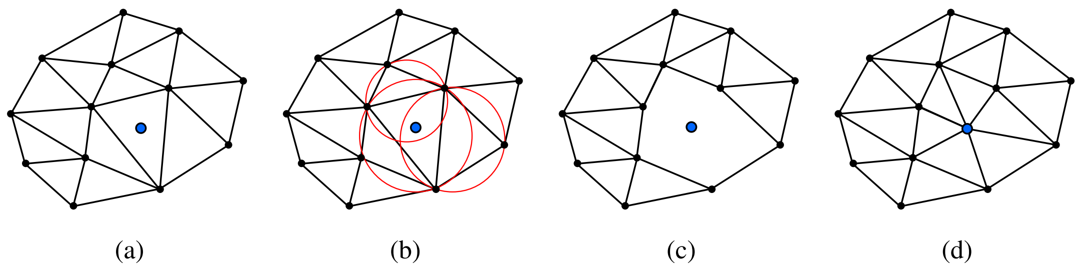


Figure 3.11: Illustration of the point insertion process in Delaunay triangulation. (a) The initial triangulation before insertion, including the new point x_i (blue node). (b) The elements whose circumcircles contain x_i are identified for removal. (c) The affected elements are deleted, forming a cavity around x_i. (d) New elements are created by connecting x_i to the cavity boundary, ensuring a valid Delaunay triangulation is maintained.

mesh. Once completed, all elements outside of the computational domain are removed – Figure 3.12(e).

At this stage, the triangulation is very coarse and often heavily skewed. It should be noted, that this stage of the volume mesh generation does not take into account prescribed spacing or discern between a mesh that is isotropic or anisotropic. Conforming to the correct spacing and anisotropic characteristics of the final mesh in the volume are introduced later during the point creation process.

Once the initial coarse tessellation is generated, it is refined to achieve the target element sizes and anisotropy. Interior points are created based on two criteria:

1. **Element-Based Insertion:** If the sizing of a whole tetrahedral element exceeds the desired size, a point is inserted at either its centroid or circumcentre.
2. **Edge-Based Insertion:** If the length of an edge surpasses the prescribed element size, a point is inserted at its midpoint.

To evaluate these criteria, the domain is first mapped into a normalised space using the transformation tensor $\mathbf{T}(\mathbf{x})$. This transformation ensures that target element sizes appear equilateral with unit size in the mapped space, simplifying the assessment of whether an element or edge requires breaking. As with surface discretisation, the

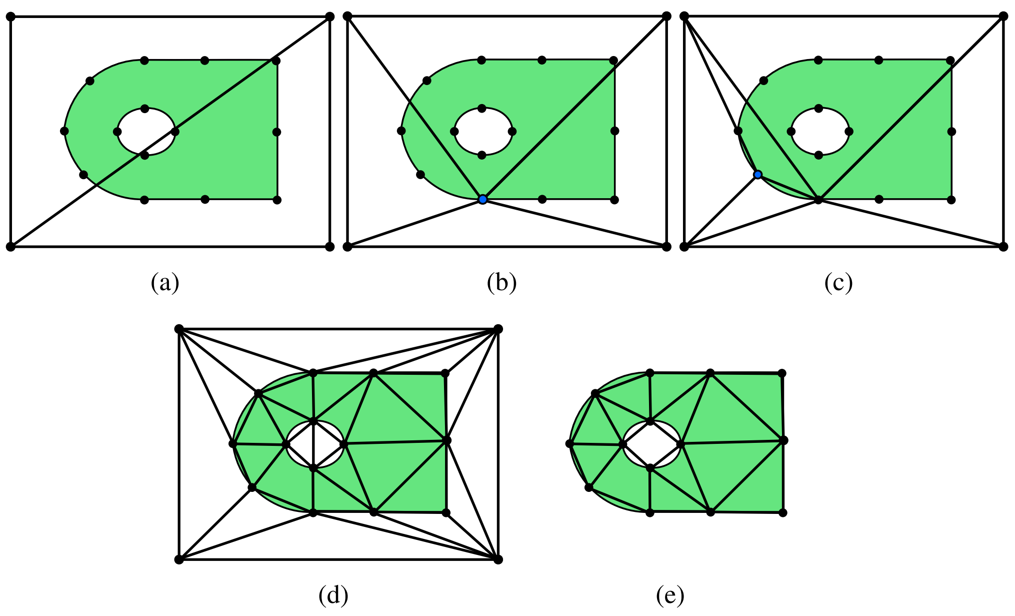


Figure 3.12: Illustration of the initial volume mesh generation process. (a) The construction of a convex hull enclosing the domain. (b & c) The sequential insertion of boundary points to form the triangulation. (d) The completed tessellation after all boundary nodes have been inserted. (e) The final application of boundary recovery techniques and removal of external elements, resulting in initial volume mesh, which remains coarse and unrefined before point creation is applied.

transformation varies continuously across the domain, meaning there is no single, exact mapping that universally defines the desired local element characteristics. To maintain local consistency in the refinement process, $\mathbf{T}$ is evaluated at the prospective new point location.

Ideally, the normalised length of an edge or the height of a tetrahedral element should be unity. If the mapped length exceeds a predefined tolerance of $\sqrt{2}$, a point is proposed for insertion to subdivide the element, thereby increasing the resolution in regions where the mesh is too coarse. However, insertion may be rejected if the resulting connectivity introduces edges that are too short. This safeguard prevents the creation of poorly shaped or ill-conditioned elements. The adaptive refinement process proceeds iteratively, proposing new points and accepting only those insertions that maintain mesh quality, until all elements conform to the prescribed mesh parameters.

Upon completion, the refined volume mesh can satisfy both isotropic and anisotropic spacing constraints while maintaining high element quality. By combining Delaunay triangulation with point insertion, this approach ensures a robust and efficient method for generating high-fidelity computational meshes, well-suited for accurate numerical simulations in CFD.

3.2.7 Mesh Enhancements

The quality of a computational mesh plays a critical role in the accuracy, stability, and efficiency of numerical simulations. Even when a mesh is generated with appropriate element sizes and distributions, imperfections such as poorly shaped elements and abrupt transitions in element shape. These deficiencies can lead to numerical instabilities, slow convergence, and inaccurate results, particularly in regions of complex geometry. To address these challenges, mesh enhancement procedures are applied as a post-processing step after the surface mesh construction (Section 3.2.5) and, subsequently, after volume mesh generation (Section 3.2.6).

Mesh enhancements aim to improve the overall quality of the mesh by enhancing element shapes, ensuring smoother transitions between adjacent elements, and maintaining alignment with the underlying geometry. This is achieved through a combination of techniques, including node smoothing, edge swapping, and edge collapse. These methods refine the mesh without altering its fundamental topology, ensuring that the simulation remains accurate and computationally efficient while avoiding the introduction of unnecessary complexity.

Node smoothing is one of the primary techniques used to reposition nodes for improved element quality. This adjustment can be performed in either the parametric space—corresponding to the underlying geometric model—or the physical space defined by the mesh. For surface meshing, smoothing is typically carried out in both the parametric and physical spaces to preserve geometric fidelity while improving element quality. In contrast, for volume meshing, smoothing is applied solely in the physical space. A common method is Laplacian smoothing, where each node is moved toward the centroid of its neighbouring nodes. This relocation reduces element skewness, improves aspect ratios, and promotes a more uniform node distribution.

Edge swapping further enhances mesh quality by modifying the connectivity between nodes. In this process, an edge shared by two adjacent elements is replaced by an alternative diagonal, forming two new elements. The decision to swap an edge is guided by the quality of the resulting elements, typically assessed using metrics such as aspect ratio, skewness, and alignment with the surface geometry. This technique often results in configurations where the midpoint of the new edge lies closer to the surface, thereby producing better-shaped elements and improving the quality of the mesh.

Edge collapse is employed to eliminate short edges and overly small elements that can lead to ill-conditioned matrices during numerical computations. This method involves collapsing an edge into a single point, effectively merging two nodes into one. The midpoint of the edge serves as an initial guess for the new node location, and the validity of the resulting mesh is evaluated in both the parametric and physical spaces. Key considerations include maintaining acceptable element areas, interior angles of triangles, and dihedral angles between adjacent tetrahedra. To prevent excessive refinement near discontinuities or curved surfaces, thresholds are imposed to avoid collapsing edges below a specified length.

By iteratively applying node smoothing, edge swapping, and edge collapse, mesh enhancements ensure that the final mesh is well-conditioned, accurately represents the underlying geometry, and supports efficient and reliable numerical simulations. This refinement process strikes a balance between resolution, element quality, and computational efficiency, ultimately enhancing the robustness of the simulation workflow.

Chapter 4

Neural Networks

The following chapter introduces the fundamental principles and methodologies underlying the neural network framework adopted in this research. It outlines the key components of neural networks relevant to this work, detailing their structure, training process, and alternative architectures. The chapter begins with a discussion on the development of neural networks, providing context for their use in computational modelling. The structure and function of feed-forward neural networks (FFNNs) are then introduced, including how information propagates through its layers.

The role of activation functions is explained, highlighting their impact on network performance and training stability. This is followed by a discussion on cost functions, specifically mean squared error (MSE) for scalar predictions and cosine similarity loss for vector-based outputs.

Optimisation methods are then examined, focusing on Stochastic Gradient Descent (SGD) and Adam, both of which are employed in training. The chapter also details backpropagation, the algorithm used to compute gradients and update model parameters efficiently.

Given the importance of proper data handling, the chapter discusses data preparation techniques, including normalisation and standardisation, ensuring consistency in training and evaluation.

This chapter establishes the necessary foundation for applying neural networks for near-optimal mesh prediction, providing the theoretical and computational basis for the models used in this research.

4.1 Machine Learning for Regression Modelling

Regression modelling is a statistical technique used to analyse and model the relationships between dependent and independent variables. It serves as a fundamental tool for predicting continuous outputs based on input features. Unlike classification, which assigns discrete labels to data points, regression approximates the underlying function that maps inputs to real-valued outputs. In scientific computing, engineering, and data-driven simulations, regression models are particularly valuable when explicit mathematical relationships between variables are unknown or too complex to derive analytically.

In recent years, machine learning has played an increasingly significant role in advancing regression analysis, providing sophisticated models capable of handling large datasets, capturing highly non-linear relationships, and managing high-dimensional data with improved accuracy and efficiency. These developments have expanded the applicability of regression techniques across various domains.

A range of regression techniques exist within machine learning, from simple linear regression to more advanced non-linear models. Traditional methods, such as polynomial regression, decision trees, and support vector regression (SVR), provide interpretable approaches to modelling relationships in data. Ensemble methods, including random forests and gradient boosting, improve predictive performance by combining multiple weak learners, effectively reducing variance and enhancing model robustness (Bishop and Clements, 2006; Hastie et al., 2009). Kriging, or Gaussian process regression

(GPR), is another widely used technique, particularly in surrogate modelling and uncertainty quantification (Rasmussen and Williams, 2005).

Among these approaches, neural networks (NNs) have gained significant attention for their flexibility and ability to capture highly non-linear functions. Neural networks, as a class of ML models, leverage multiple layers of interconnected neurons to learn complex mappings between inputs and outputs. This flexibility makes them particularly effective for high-dimensional regression problems where conventional methods may struggle.

4.2 Neural Networks Overview

Neural networks (NNs) are a class of machine learning models designed to approximate complex functions by iteratively adjusting their parameters to minimise error. They operate on the principle of transforming inputs through a series of weighted computations, applying non-linear functions, and refining their predictions using optimisation techniques.

The concept of artificial neurons dates back to the work of McCulloch and Pitts in the 1940s (McCulloch and Pitts, 1943), where they introduced a mathematical model capable of performing logical operations. This early work demonstrated that networks of simple threshold-based units could, in principle, compute any function. In 1958, the perceptron model developed by Rosenblatt (Rosenblatt, 1958) extended this idea by introducing a learning algorithm capable of adjusting connection weights based on input-output discrepancies. However, the perceptrons were limited to solving linearly separable problems, as shown by Minsky and Papert (Minsky and Papert, 1987), leading to a decline in research interest.

The resurgence of neural networks in the 1980s was driven by the introduction of multi-layer perceptrons (MLPs) and the backpropagation algorithm, popularised by Rumelhart, Hinton, and Williams (Rumelhart et al., 1986). Backpropagation enabled the efficient training of networks with multiple layers, allowing for the approximation of non-linear functions by iteratively adjusting weights to minimise the error between predicted and target values.

At a fundamental level, neural networks function as follows: given an input vector, the network applies a series of transformations through hidden layers, producing an output. This output is then compared to the expected result, and a cost function quantifies the error. The learning process involves computing the gradient of this error with respect to the network parameters (weights and biases). Using an optimisation algorithm, the parameters are updated by moving in the direction that minimises the error. This iterative process continues until the network reaches a parameter configuration that yields satisfactory performance, typically measured by the stabilisation of the loss on a validation set or the fulfilment of early stopping criteria.

This continuous parameter adjustment allows neural networks to learn complex mappings between inputs and outputs. With advances in computational power and the availability of large datasets, neural networks – particularly deep learning models, have become powerful tools across various domains.

4.2.1 Feed-forward neural networks

Neural networks (NNs) are a class of computational models inspired by the structure and function of biological neural networks in the brain. In biological systems, neurons receive input signals through dendrites, process the information, and propagate output signals through axons to other neurons. This interconnected system of neurons enables complex pattern recognition, learning, and decision-making capabilities. Neural networks mimic this behaviour by defining layers of interconnected artificial neurons, each performing a weighted sum of its inputs followed by a non-linear activation function.

A specific type of NN, and the one employed in this work, is the Feed-Forward Neural Network (FFNN). In an FFNN, information flows in a single direction, from the input

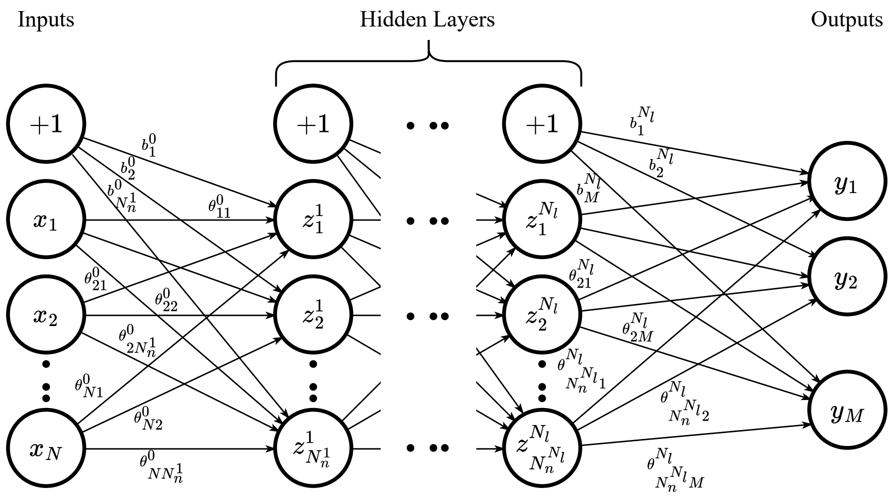


Figure 4.1: Schematic of a feed-forward multi-layer perceptron (MLP) architecture.

layer through one or more hidden layers to the output layer, with no cycles or feedback connections between nodes. This directed flow, combined with the layered structure, makes FFNNs particularly well-suited for supervised learning tasks.

Figure 4.1 provides a schematic representation of a typical feed-forward multi-layer perceptron (MLP), a well-known subclass of FFNN. The architecture consists of an input layer, a number of hidden layers, and an output layer.

The input layer contains N nodes, representing the individual parameters of the input vector $(x_1, x_2, \dots, x_N)$. Between the input and output layers lie the hidden layers, shown in the central portion of Figure 4.1. A feed-forward neural network may consist of multiple hidden layers, denoted N_l, where each hidden layer may contain a different number of neurons. In layer c, the number of neurons is denoted N_n^c. In the figure, the first hidden layer contains N_n^1 neurons, denoted $(z_1^1, z_2^1, \dots, z_{N_n^1}^1)$, while the final hidden layer contains $N_n^{N_l}$ neurons, denoted $(z_1^{N_l}, z_2^{N_l}, \dots, z_{N_n^{N_l}}^{N_l})$. The hidden layers play a central role in the network's ability to approximate complex functions, as they allow successive compositions of non-linear mappings to represent intricate dependencies between inputs and outputs. Lastly, the output layer, shown on the right side of Figure 4.1, contains M output nodes, where M is task dependent.

As illustrated in Figure 4.1, each layer also includes a bias term, represented by a constant neuron with a fixed output of $+1$. The primary purpose of the bias term is to allow the activation function to shift independently of the weighted sum of inputs. Without a bias term, the weighted sum would always pass through the origin when the inputs are zero, limiting the flexibility of the network. By introducing a bias, the neuron can learn relationships that do not necessarily pass through the origin, enabling the network to model a broader class of functions.

Each neuron in the network receives inputs from all neurons in the preceding layer (including the bias), resulting in a fully connected architecture. Each connection is associated with a weight, denoted by θ and b, which are adjusted during training to minimise the error between the predicted and target outputs. Specifically, θ^c_{ab} denotes the weight connecting the a-th neuron of the c-th layer to the b-th neuron of the next layer, and b^c_a denotes the bias from the a-th layer to the c-th neuron of the next layer.

4.2.2 Forward Propagation and Activation Functions

In a feed-forward neural network, information flows sequentially from the input layer towards the output layer through the intermediate hidden layers. This process, referred to as forward propagation, involves computing the output of each neuron as a weighted sum of the outputs from the previous layer, followed by the application of an activation function.

For a neuron j in layer $l + 1$, the output value, denoted z_j^{l+1}, is computed using the outputs of the neurons in layer l, the weights of the connections between layers, and the bias term associated with the neuron. Mathematically, this can be written as

$$z_j^{l+1} = F^{l+1}\left(\sum_{i=1}^{N_n^l} \theta^l_{ji} z^l_i + b^l_j\right), \tag{4.1}$$

where F^{l+1} denotes the activation function applied in layer $l+1$, θ^l_{ji} denotes the weight connecting neuron i in layer l to neuron j in layer $l+1$, z_i^l represents the output of neuron i in layer l, and b_j^l is the bias term for neuron j in layer l.

The activation functions F^k introduce non-linearity into the network, which is essential to allow the network to approximate complex relationships between inputs and outputs. Without non-linearity, the entire network would collapse into a single linear transformation, regardless of the number of neurons or layers, severely limiting its ability. Different layers can apply different activation functions depending on the requirements of the task or the characteristics of the data. Below are some commonly used activation functions along with their corresponding derivatives:

- Linear:
$$L(x) = x, \quad L'(x) = 1, \tag{4.2}$$

- Rectified Linear Unit (ReLU):
$$R(x) = \max(0, x), \quad R'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0, \end{cases} \tag{4.3}$$

- Sigmoid:
$$S(x) = \frac{1}{1 + e^{-x}}, \quad S'(x) = S(x)(1 - S(x)), \tag{4.4}$$

- Hyperbolic tangent (tanh):
$$T(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad T'(x) = 1 - T(x)^2. \tag{4.5}$$

These functions, along with their derivatives, are illustrated in Figure 4.2. The choice of activation function plays a crucial role in determining both the performance of the network and its training efficiency. Each hidden layer is free to employ different activation functions. In this work, which focuses on regression modelling, the output layer employs a linear activation, while classification tasks typically use sigmoid or softmax activations.

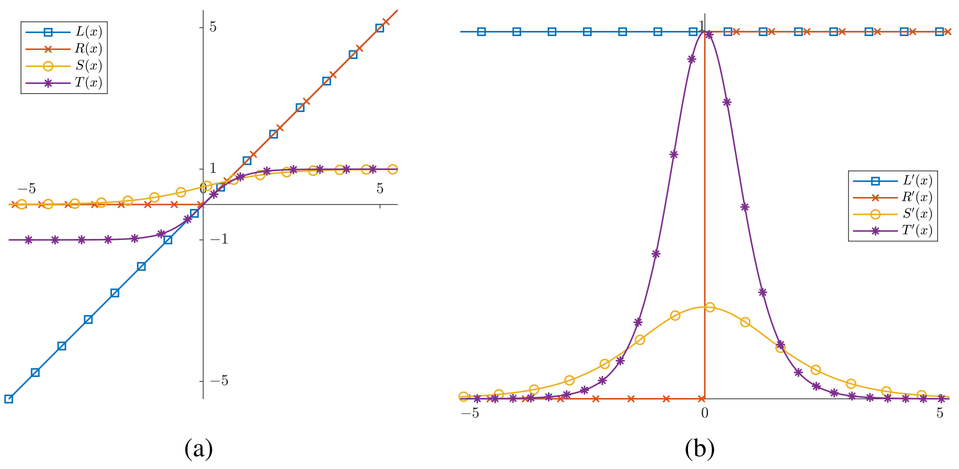


Figure 4.2: (a) Common activation functions used in neural networks, including the Linear function $L(x)$, Rectified Linear Unit (ReLU) $R(x)$, Logistic Sigmoid $S(x)$, and Hyperbolic Tangent $T(x)$. (b) The corresponding derivatives of these activation functions, illustrating their behaviour and sensitivity to input changes.

The historical development of activation functions reflects the evolving understanding of how to best introduce non-linearity while maintaining stable and efficient training. Early neural networks primarily employed the sigmoid function due to its smooth, bounded output, which made it well-suited for probabilistic interpretation in classification tasks. Similarly, the hyperbolic tangent function, which maps to $[-1, 1]$ rather than $[0, 1]$, became popular due to its symmetric output range, which was found to improve training convergence in some cases.

Despite their initial popularity, both the sigmoid and hyperbolic tangent activation functions suffer from the vanishing gradient problem, where gradients become extremely small for large positive or negative inputs. This significantly hampers weight updates during training, particularly in deep networks, slowing convergence or leading to stagnation (Aggarwal, 2023; Ian Goodfellow and Courville, 2016).

The introduction of the ReLU activation function marked a substantial shift in neural network design. Its computational simplicity and non-saturating gradient for positive inputs helped mitigate the vanishing gradient issue, particularly in deep networks. These advantages made ReLU the dominant choice in modern deep learning. However, ReLU is not without limitations, as it can suffer from the dying ReLU problem, where neurons output zero for all inputs, effectively removing them from the network (Douglas and Yu, 2018).

The forward propagation process, combining weighted sums, bias terms, and activation functions, allows the network to construct hierarchical, abstract representations of the input data, enabling the final output layer to make predictions based on these progressively refined features. The choice of activation function directly influences the ability of the network to learn complex mappings, its convergence behaviour, and its overall performance.

4.2.3 Parameter Initialisation

Before training can begin, the weights and biases of the network must be initialised. The initialisation strategy has a significant impact on the stability and efficiency of the learning process. Poorly chosen initial values can result in vanishing or exploding gradients, slow convergence, or failure to escape suboptimal regions of the loss landscape. Effective initialisation methods are thus essential to ensure that training proceeds reliably and that the network can learn meaningful patterns from the data.

In this work, the initialisation strategy is selected based on the activation function employed in each layer. For layers using sigmoid, tanh or linear activations, weights are initialised using the Glorot uniform initialiser, also referred to as the Xavier uniform initialiser (Glorot and Bengio, 2010). This method maintains consistent variance across layers by drawing initial weights from a uniform distribution bounded by

$$\theta^l_{jk} \sim \mathcal{U}\left(-\sqrt{\frac{6}{N^l_n + N^{l+1}_n}}, \sqrt{\frac{6}{N^l_n + N^{l+1}_n}}\right). \tag{4.6}$$

This helps prevent early-stage saturation in non-linear activations and supports stable gradient propagation.

For layers utilising ReLU activations, weights are initialised using the He normal initialiser (He et al., 2015), which draws samples from a zero-mean normal distribution with variance scaled by the number of inputs into the layer

$$\theta^l_{jk} \sim \mathcal{N}\left(0,\ \frac{2}{N^l_n}\right). \tag{4.7}$$

This choice is particularly well-suited for ReLU-based networks, compensating for the asymmetric activation behaviour by maintaining the magnitude of activations and gradients throughout the depth of the network.

All bias terms are initialised to zero, a standard and computationally efficient practice. As biases do not depend on input magnitudes, zero initialisation does not interfere with the symmetry-breaking essential to learning dynamics.

To minimise the impact of initialisation and to promote training robustness, different random seeds are used to generate multiple realisations of initial weight values. While the initialiser defines the statistical distribution, the seed determines the specific values sampled. This allows for comparative training runs and ensures that final model performance is not biased by a single random starting point.

4.2.4 Training the Neural Network

Once the architecture of the neural network has been defined and the parameters have been initialised, the next step is to train the network by adjusting its weights and biases to minimise the discrepancy between the predicted and target outputs. This requires the formulation of a cost function, which quantitatively measures the performance of the model for a given set of training data.

Let N_{tr} denote the number of training cases. Each training case consists of an input vector, $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}^T$, containing N input features, and a corresponding output vector, $\mathbf{y} = \{y_1, y_2, \ldots, y_M\}^T$, containing M target outputs.

In this work, when the neural network is used to predict scalar quantities, M represents the number of scalar outputs. When the network is used to predict vector fields, M represents a flattened form of the vector field, with the field composed of N_{vec} vectors, each with n_{dim} components. The output vector $\mathbf{y}$ is therefore a concatenation of these N_{vec} vectors, resulting in $M = N_{\text{vec}} \times n_{\text{dim}}$.

The neural network maps the input vector to a predicted output vector $\hat{\mathbf{y}}$ through the process of forward propagation. To evaluate the performance of the network, a cost function, denoted $C(\boldsymbol{\theta})$, is used, where $\boldsymbol{\theta}$ represents the set of all trainable parameters (weights and biases) in the network. The cost function measures the discrepancy between the network predictions, $\hat{\mathbf{y}}(\mathbf{x}, \boldsymbol{\theta})$ and the true outputs for a single given case. To compute a global loss, the average loss across all training cases can be taken.

In this work, two cost functions are employed, depending on the nature of the output data: the mean squared error (MSE) and the cosine similarity loss.

4.2.4.1 Mean Squared Error

The Mean Squared Error (MSE) is used when the network is predicting a set of scalar outputs. For a single training case, the MSE is defined as

$$C(\boldsymbol{\theta}) = \frac{1}{M} \sum_{i=1}^{M} \left(\hat{y}_i(\mathbf{x}, \boldsymbol{\theta}) - y_i(\mathbf{x})\right)^2, \tag{4.8}$$

where $\hat{y}_i$ and y_i denote the predicted and true values of the i-th output, respectively. The MSE penalises larger deviations more heavily than smaller ones, encouraging the network to learn an accurate mapping across all output dimensions.

The squared term ensures the loss function is smooth and differentiable everywhere, unlike the absolute value which introduces a discontinuity in the gradient at zero. Differentiability is crucial for gradient-based optimisation algorithms, such as stochastic gradient descent, which rely on computing partial derivatives of the loss with respect to the network parameters to update them during training. Using a differentiable loss function like MSE guarantees that the gradient is well-defined and continuous, enabling stable and efficient convergence of the training process.

4.2.4.2 Cosine Similarity Loss

When the network is used to predict a vector field, such as in Section 7.1.4, the cosine similarity loss is employed. In these cases, the output at each training case consists of N_{vec} vectors, each with n_{dim} components. The predicted and target outputs are thus structured as

$$\hat{\mathbf{y}} = \begin{bmatrix} \hat{\mathbf{y}}_1 & \hat{\mathbf{y}}_2 & \dots & \hat{\mathbf{y}}_{N_{\text{vec}}} \end{bmatrix}^T \in \mathbb{R}^M$$

where $M = N_{\text{vec}} \times n_{\text{dim}}$, and $\hat{\mathbf{y}}_i \in \mathbb{R}^{n_{\text{dim}}}$ denotes the predicted vector at the i-th location. In this case, the cosine similarity loss for a single case is defined as

$$C(\boldsymbol{\theta}) = \frac{1}{N_{\text{vec}}} \sum_{i=1}^{N_{\text{vec}}} \left(1 - \frac{\hat{\mathbf{y}}_i(\mathbf{x}, \boldsymbol{\theta}) \cdot \mathbf{y}_i(\mathbf{x})}{\|\hat{\mathbf{y}}_i(\mathbf{x}, \boldsymbol{\theta})\| \|\mathbf{y}_i(\mathbf{x})\|} \right), \tag{4.9}$$

where $\hat{\mathbf{y}}_i$ and $\mathbf{y}_i$ denote the predicted and true vectors at the i-th location, respectively. This formulation ensures that the network learns to predict vectors with the correct orientation in n_{dim} space. In this work, the target vectors $\mathbf{y}_i$ are always unit vectors, meaning $\|\mathbf{y}_i\| = 1$. Consequently, their magnitude is omitted from the loss calculation.

4.2.5 Overfitting and Validation Strategy

In machine learning tasks, particularly when training neural networks, controlling overfitting is essential to ensure that models generalise beyond the training data. Overfitting occurs when the model learns to reproduce the training outputs with high accuracy while failing to capture the underlying relationship between inputs and outputs, leading to poor predictive performance on unseen data.

A standard strategy to monitor and reduce overfitting involves partitioning the available data into three subsets: training, validation, and test sets. The validation set allows one to evaluate performance during training, thereby informing decisions such as when to stop training or how to adjust hyperparameters. However, in the context of this work, each data point corresponds to an expensive high-fidelity CFD simulation, meaning the overall number of available examples is limited. Allocating a dedicated validation subset would significantly reduce the amount of data available for training and is therefore avoided.

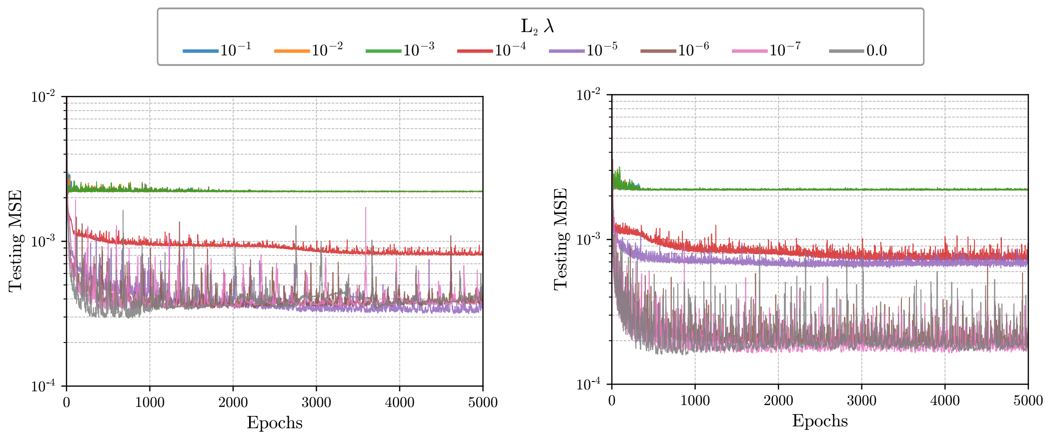
Instead, in this work, the dataset is divided into two subsets only: a training set and a test set for final evaluation. This constraint necessitates alternative mechanisms to avoid overfitting.

Although regularisation techniques such as L_2 weight decay, dropout, and noise injection are often employed to suppress overfitting, their utility diminishes in severely data-constrained settings (Bengio, 2016). Furthermore, these methods introduce additional hyperparameters – such as the regularisation coefficient or dropout rate, which must be carefully tuned. Improper settings may either over-constrain the network, resulting in underfitting, or fail to suppress overfitting effectively, thereby compromising model accuracy. Conducting exhaustive hyperparameter sweeps under such conditions imposes a significant computational burden and is therefore impractical.

To examine the impact of L_2 weight regularisation, a series of experiments were conducted across a range of λ values and two network sizes. The smaller model comprises two hidden layers with 50 neurons each, while the larger model consists of five hidden layers with 200 neurons per layer. In L_2 regularisation, or weight decay, the standard loss function is augmented with a penalty term proportional to the squared norm of the model weights. The modified cost function takes the form,

$$C(\boldsymbol{\theta}) = C_{\text{data}}(\boldsymbol{\theta}) + \lambda \sum \theta_i^2, \tag{4.10}$$

where C_{data} denotes the original data loss (e.g., mean squared error), θ_i are the trainable weights, and λ controls the strength of the regularisation. This formulation discourages



(a) Small Model (b) Large Model

Figure 4.3: Effect of varying L_2 regularisation strength λ on testing MSE during training for (a) a small model and (b) a large model.

large weights and promotes simpler, more generalisable models.

Figure 4.3 shows the effect of varying λ for both architectures over 5000 training epochs. For both models, the lack of regularisation leads to a rapid and initially deeper reduction in the test error. In the smaller network, the absence of regularisation leads to a rise in test error after initial convergence, indicating overfitting. Applying L_2 regularisation markedly improves generalisation, with $\lambda = 10^{-5}$ yielding the best performance. In contrast, the larger model with $\lambda = 10^{-5}$ yields a noticeable worsening in the performance of the model.

These observations highlight the difficulty of selecting a single hyperparameter value to reduce overfitting, although an exhaustive hyperparameter optimisation would likely yield improved performance, the associated computational cost would be incompatible with the efficiency objectives of this work. Instead, a heuristic approach is adopted, using a small number of exploratory experiments to guide the choice of training epochs. By analysing the training curves across different configurations, like those shown in Figure 4.3, an appropriate number of epochs for all models can be selected to achieve convergence and minimise overfitting – striking a practical balance between simplicity, efficiency, and generalisation.

4.3 Optimisers

In machine learning, the accuracy of a model is intrinsically linked to its loss function, which quantifies how well its predictions align with the true target values. The loss function provides feedback that guides parameter adjustments to minimise errors, making loss minimisation the central objective in model training.

When training a model, optimisation algorithms iteratively update model parameters to reduce the loss function. These algorithms determine the gradient of the cost function with respect to each parameter and adjust them accordingly. Lower loss values indicate improved predictions, leading to a more accurate model.

However, minimising the loss function is not always straightforward. Challenges such as overfitting, local minima, and vanishing or exploding gradients can hinder convergence and affect model performance. Consequently, selecting an appropriate optimiser plays a crucial role in effectively navigating these difficulties and achieving optimal performance.

4.3.1 Gradient Descent Optimisation

Gradient descent is one of the most widely used optimisation algorithms in machine learning, providing an iterative method for minimising the cost function and improving model accuracy. At each step, the model parameters are updated by moving in the direction of the negative gradient of the loss function, ensuring a gradual reduction in error. The basic form of gradient descent updates the parameters according to

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \frac{\partial C(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}, \tag{4.11}$$

where $\boldsymbol{\theta}$ represents the model parameters, η is the learning rate, and $C(\boldsymbol{\theta})$ is the loss function. The learning rate controls the step size of each update, with smaller values leading to slower but more stable convergence, while larger values risk overshooting the minimum.

There are several variations of gradient descent, each differing in how the gradient is computed and applied to update the model parameters (Aggarwal, 2023; Ian Goodfellow and Courville, 2016; Ruder, 2016).

Stochastic Gradient Descent (SGD) updates the model parameters after computing the gradient using a single randomly chosen data point. Instead of waiting to process the entire dataset, each training example contributes to a parameter update. At the beginning of each epoch, the training data is randomly shuffled, and individual samples are then processed in sequence according to this new ordering. While this makes SGD computationally efficient and enables it to escape sharp local minima, it also introduces significant noise into the optimisation process, causing parameter updates to fluctuate rather than converge smoothly. As a result, the path towards the minimum can be erratic, and convergence may require careful tuning of the learning rate.

Batch Gradient Descent computes the gradient using the entire dataset before performing an update. This ensures that each step is in the precise direction of the steepest descent for the full loss function, leading to stable and consistent convergence. However, for large datasets, computing the gradient over all samples at every iteration can be prohibitively expensive in terms of both memory and computation time.

Mini-Batch Gradient Descent offers a balance between the two approaches. Instead of updating after every single data point (as in SGD) or computing for the entire dataset (as in batch gradient descent), mini-batch gradient descent updates the parameters using a randomly selected subset of the training data, known as a batch. Like in SGD, the training dataset is randomly shuffled once at the start of each epoch, and batches are then drawn sequentially from this shuffled ordering. The update rule for mini-batch gradient descent is

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \frac{1}{N_b} \sum_{k=1}^{N_b} \frac{\partial C_k(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}, \tag{4.12}$$

where N_b is the batch size, and the sum accounts for all training samples in the batch.

Mini-batch gradient descent provides a balance between the stability of full-batch gradient descent and the efficiency of stochastic gradient descent. Computing updates over a subset of the training data reduces the variance of individual updates in stochastic gradient descent while avoiding the high computational cost of full-batch methods. Additionally, the moderate level of noise introduced by mini-batch updates can help the optimiser navigate non-convex loss landscapes more effectively, improving generalisation and preventing convergence to sharp local minima.

Mini-batch gradient descent, however, remains highly sensitive to the choice of hyperparameters, particularly the learning rate. If the learning rate is too large, the optimiser may overshoot the minimum, leading to divergence or oscillatory behaviour. Conversely, if it is too small, convergence can be excessively slow, preventing the model from learning efficiently. Proper tuning of the learning rate is therefore essential to ensure stable and effective training.

In this thesis, the term *Stochastic Gradient Descent (SGD)* will be used to refer specifically to mini-batch gradient descent, a common convention in machine learning literature. It is important to note that, despite this naming convention, true SGD – which updates after every individual training sample, is rarely used in practice due to its instability. Instead, the term often refers to mini-batch gradient descent with data shuffled once per epoch and processed in batch-sized increments. This distinction is important, as true SGD (updating after every sample) is rarely used in practice due to its instability, while mini-batch gradient descent provides a more practical and efficient approach to training neural networks.

4.3.2 Adam Optimiser

Neural network researchers have long recognised that the learning rate is one of the most challenging hyperparameters to tune, as it significantly impacts model performance and convergence. The choice of learning rate can be highly sensitive; if set too high, the optimiser may overshoot the minimum, leading to divergence, whereas a learning rate that is too low can result in excessively slow convergence.

To address these challenges, algorithms with adaptive learning rates were developed. The momentum algorithm (Polyak, 1964) was an early improvement, helping to smooth updates and accelerate convergence in the presence of noisy gradients. Subsequent advancements led to methods such as AdaGrad (Duchi et al., 2011), which adapted the learning rate for each parameter based on past gradients, and RMSProp, which further refined this approach by introducing an exponentially decaying average of squared gradients (Ian Goodfellow and Courville, 2016).

Adam (Adaptive Moment Estimation) builds upon these ideas and can be seen as a variant that effectively combines elements of RMSProp and momentum (Kingma and Ba, 2014). Introduced by Kingma and Ba in 2014, Adam has become one of the most widely used optimisation algorithms in deep learning due to its efficiency and robustness across a wide range of tasks.

Adam maintains two moving averages for each parameter: the first-moment estimate, m_t, which tracks the gradient, and the second-moment estimate, v_t, which tracks the squared gradient. These moving averages are used to adjust the learning rate dynamically for each parameter, ensuring more stable and adaptive updates. At each optimisation step t, after computing the gradient of the cost function with respect to the model parameters, the first-moment estimate is updated as

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\frac{1}{N_b}\sum_{k=1}^{N_b}\frac{\partial C_k(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}, \tag{4.13}$$

while the second-moment estimate is computed as

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\frac{1}{N_b}\sum_{k=1}^{N_b}\left(\frac{\partial C_k(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right)^2. \tag{4.14}$$

Here, β_1 and β_2 are decay rates for the moving averages, typically set to 0.9 and 0.999, respectively, and N_b is the batch size used to compute the gradients at each step.

Since these moving averages are initialised to zero at $t = 0$, they are biased toward zero in the early iterations. To correct for this bias, Adam applies bias correction, computing

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t}, \quad \text{and} \quad \hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_2^t}. \tag{4.15}$$

These bias-corrected estimates approximate the true first- and second-moment values, particularly in the initial training iterations when t is small.

Once the corrected moment estimates are obtained, the parameters are updated using the rule

$$\boldsymbol{\theta}_t := \boldsymbol{\theta}_{t-1} - \frac{\eta}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} \hat{\mathbf{m}}_t. \tag{4.16}$$

The learning rate η controls the step size of each update, while ϵ is a small constant (typically 10^{-8}) added for numerical stability, preventing division by zero. With each optimisation step, t increments, continuously refining the moving averages and adjusting the parameter updates accordingly.

Adam provides an adaptive learning rate for each parameter, which makes it highly effective across a wide range of applications. It converges quickly, efficiently handles saddle points, and requires minimal memory, making it well-suited for training efficiently. The algorithm incorporates the benefits of momentum, which accelerates convergence while simultaneously stabilising updates through its adaptive learning rate.

Despite these advantages, Adam is not always optimal. It may fail to converge to the best solution, a phenomenon known as the Adam convergence problem, where the optimiser appears to have converged but settles at a suboptimal point. Additionally, its adaptability can lead to overfitting, particularly in high-variance gradient environments. In certain cases, Stochastic Gradient Descent has been found to provide better generalisation, particularly in deep learning applications.

Adam is a powerful and versatile optimisation algorithm that has become a staple in many machine-learning applications. Its ability to adapt learning rates automatically and its robustness to different problem types make it a popular choice for many machine learning tasks.

4.4 Backward Propagation

All optimisation algorithms discussed in the previous section require the computation of the gradient of the cost function with respect to the network parameters, collectively denoted as $\boldsymbol{\theta}$. These gradients provide the necessary information to update the weights and biases in a manner that reduces the cost function, thereby improving the predictions given by the model. The process of computing these gradients efficiently across all layers of a neural network is known as backward propagation or backpropagation (Rumelhart et al., 1986).

The backpropagation algorithm was developed to address the challenge of efficiently computing gradients in multi-layer neural networks. Before its formalisation, training deep networks was computationally prohibitive due to the need for explicit differentiation of each network parameter, which was infeasible for large-scale models. Early neural networks, such as the Perceptron, lacked a structured approach to weight updates in multi-layer architectures, leading to significant limitations in their learning capabilities (Rosenblatt, 1958).

The modern formulation of backpropagation was promoted by Rumelhart, Hinton, and Williams, who demonstrated how the chain rule of differentiation could be applied efficiently in layered networks (Rumelhart et al., 1986; Ian Goodfellow and Courville, 2016). Their work built upon earlier principles of automatic differentiation and gradient-based learning, enabling the systematic computation of weight updates across multiple layers. This breakthrough made it feasible to train deep multi-layer perceptrons (MLPs) and became the foundation for modern deep learning. Although the chain rule had been known in mathematical optimisation for decades, their work provided an explicit and computationally efficient algorithm for applying it to neural networks, revolutionising the field of artificial intelligence.

4.4.1 Mathematical Formulation of Backpropagation

Backpropagation is based on the chain rule of differentiation, which allows gradients to be propagated backwards through the network. For the neural network formulation given in Equation (4.1), the objective is to compute the gradient of the cost function $C(\boldsymbol{\theta})$ with respect to the network parameters—specifically, the weights θ^l_{ji} and biases b^l_j.

Applying the chain rule, the gradient with respect to any weight in layer l is

$$\frac{\partial C}{\partial \theta^l_{ji}} = \frac{\partial C}{\partial z^{l+1}_j} \frac{\partial z^{l+1}_j}{\partial \theta^l_{ji}}. \tag{4.17}$$

Expanding the second term, the partial derivative of the neuron output with respect to its weight is

$$\frac{\partial z^{l+1}_j}{\partial \theta^l_{ji}} = (F^{l+1})' \left(\sum_{i=1}^{N^l_n} \theta^l_{ji} z^l_i + b^l_j \right) z^l_i. \tag{4.18}$$

Defining the error term at neuron j in layer $l+1$ as

$$\delta^{l+1}_j = \frac{\partial C}{\partial z^{l+1}_j} (F^{l+1})' \left(\sum_{i=1}^{N^l_n} \theta^l_{ji} z^l_i + b^l_j \right), \tag{4.19}$$

the weight gradient simplifies to

$$\frac{\partial C}{\partial \theta^l_{ji}} = \delta^{l+1}_j z^l_i. \tag{4.20}$$

To propagate gradients backward, the error term at layer l must be expressed in terms of the error term at layer $l+1$. Differentiating the cost function with respect to the activations in layer l gives

$$\frac{\partial C}{\partial z_i^l} = \sum_{j=1}^{N_n^{l+1}} \frac{\partial C}{\partial z_j^{l+1}} \frac{\partial z_j^{l+1}}{\partial z_i^l}. \tag{4.21}$$

Expanding the second term,

$$\frac{\partial z_j^{l+1}}{\partial z_i^l} = (F^{l+1})' \left(\sum_{i=1}^{N_n^l} \theta_{ji}^l z_i^l + b_j^l \right) \theta_{ji}^l, \tag{4.22}$$

leads to the recursive error relation,

$$\delta_i^l = (F^l)'(z_i^l) \sum_{j=1}^{N_n^{l+1}} \theta_{ji}^l \delta_j^{l+1}. \tag{4.23}$$

This recursive equation allows error terms to be computed layer by layer, starting from the output layer and propagating backwards through the network. Once gradients of the cost function with respect to the network parameters are obtained, they are used to update the weights and biases using an optimisation algorithm such as gradient descent or its variants.

Modern NN frameworks such as TensorFlow and PyTorch automate backpropagation using *automatic differentiation*, eliminating the need for manual gradient computation. During the forward pass, these frameworks construct a computational graph, recording inputs, outputs, and derivatives for each operation. Gradients are then computed using *reverse-mode differentiation*, which propagates errors from the output layer back through the network.

Reverse-mode differentiation is highly efficient for neural networks, as it allows gradients of all parameters to be computed in a single backward pass, keeping computational costs manageable even in deep architectures. By handling gradient computation and weight updates automatically, they simplify neural network implementation while ensuring computational efficiency.

4.5 Data preparation

Before training a neural network, it is essential to preprocess the data to ensure that the model can learn effectively. The raw input data may contain features with vastly different scales, distributions, or magnitudes, which can negatively impact training stability and convergence. Proper data preparation ensures that all input and output parameters are appropriately scaled, preventing numerical instability and improving optimisation efficiency.

In this work, three approaches to data preparation are considered: using raw data without transformation, normalisation, and standardisation. The choice of transformation depends on the nature of the data and the specific requirements of the neural network model.

When scaling the data, all transformations are computed using statistics derived only from the training dataset. The same transformations are then applied to the testing dataset to ensure consistency and prevent information leakage. Input parameters are scaled independently, preserving their individual distributions and relative variations. Similarly, output parameters are scaled independently, ensuring that the network learns to predict each output variable without biases introduced by differing magnitudes.

Normalisation rescales the data to a fixed range, typically between zero and one, and is given by

$$\hat{u} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}, \tag{4.24}$$

where x is the original value, $x_{\min}$ and $x_{\max}$ are the minimum and maximum values of the feature in the training set, and x' is the normalised value. Normalisation is a useful technique when features have a bounded range and no strong outliers. Normalisation ensures that the transformed training data lies strictly within the range $[0, 1]$. However, it should be noted that since the normalisation parameters are determined from the training data, any test data points that fall outside the original range of $[x_{\min}, x_{\max}]$ will be mapped to values outside $[0, 1]$.

Standardisation transforms the data to have zero mean and unit variance and is computed as

$$\hat{u} = \frac{x - \mu}{\sigma}, \tag{4.25}$$

where μ is the mean and σ is the standard deviation of the feature in the training set. Unlike normalisation, standardisation does not constrain the values to a fixed range but ensures that most values lie within a few standard deviations of zero.

Certain input parameters exhibit extreme variations in magnitude across different regions of the domain. For example, in this work, the spacing parameter can vary by several orders of magnitude across the domain. Directly comparing such values would heavily bias predictions towards the larger spacing values, which are often less critical. To mitigate this, a logarithmic transformation is applied to parameters with large ranges *before* any normalisation or standardisation

$$\hat{u} = \log_{10}(x), \quad x > 0. \tag{4.26}$$

Applying the logarithm first ensures that subsequent scaling transformations operate on a more balanced range of values, preventing the dominance of larger magnitudes. This transformation compresses large values while expanding smaller ones, leading to a more uniform distribution and improving the stability of the learning process.

4.6 Neural Network Evaluation

Evaluating the performance of a neural network requires assessing how well the predicted outputs match the target values. Ideally, for a given test case, the accuracy of the neural network would be measured by directly comparing the predicted mesh with the target mesh in a quantitative manner. However, this approach is computationally impractical and becomes increasingly complex when multiple neural network models are predicting different features simultaneously.

To overcome these challenges, statistical measures are used to evaluate the network's performance efficiently. Rather than comparing entire mesh structures explicitly, the predicted and target values for individual features are compared on a case-by-case basis using well-established error metrics. In this work, two primary evaluation metrics are employed: the coefficient of determination (R^2 score) and the Mean Absolute Error (MAE). These metrics provide an efficient proxy for approximating how closely the predicted and target spacing fields align.

Performance evaluation is conducted after the predicted outputs have been denormalised to ensure that errors are assessed in the original scale of the data. However, for parameters that underwent a logarithmic transformation, such as spacing, the evaluation is performed in the log-transformed space. This ensures consistency in error measurement and prevents the results from being biased by extreme variations in magnitude.

The R^2 score measures how well the predicted values explain the variance in the target values (Glantz et al., 2015). It is defined as

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}, \tag{4.27}$$

where y_i and $\hat{y}_i$ denote the true and predicted values for the i-th test case, respectively, and $\bar{y}$ is the mean of the true values. The numerator represents the sum of squared errors (SSE), which quantifies the total squared differences between predictions and true values. The denominator represents the total sum of squares (SST), which measures the total variance present in the target data.

An R^2 score of 1 indicates a perfect fit, meaning the model predictions exactly match the target values. A score of 0 suggests that the model performs no better than simply predicting the mean of the target values, meaning it fails to capture any meaningful relationship. Negative values indicate that the model introduces more error than a naïve mean-based prediction.

The Mean Absolute Error (MAE) provides an intuitive measure of the average absolute difference between predicted and target values. It is defined as

$$\text{MAE} = \frac{1}{N_{te}} \sum_{i=1}^{N} |y_i - \hat{y}_i|, \tag{4.28}$$

where N_{te} is the total number of test cases. Unlike the R^2 score, which is scale-invariant, the MAE retains the original units of the target variable, making it useful for interpreting the magnitude of prediction errors directly. Lower MAE values indicate better predictive performance, as they correspond to smaller average deviations between predicted and actual values.

These statistical metrics serve as effective approximations for evaluating how closely the predicted and target spacing fields match. While they do not capture the full complexity of mesh comparisons, they provide a computationally efficient and interpretable means of assessing model performance across different cases.

Chapter 5

Predicting the Near-optimal Mesh using Sources

This chapter presents a methodology to predict near-optimal mesh spacing functions using neural networks, as outlined in the paper ”*Meshing using neural networks for improving the efficiency of computer modelling*,” published in *Engineering with Computers* (Lock et al., 2023).

The strategy consists of four key stages. First, a technique is introduced to generate point sources from existing CFD solutions. These point sources capture the local mesh requirements derived from the solution field via Hessian-based error estimation. A continuous spacing function is then constructed to represent these discrete values. This process is repeated for each solution in the dataset, resulting in multiple sets of point sources.

To enable neural network training, a method is proposed to combine these case-specific sources into a single global set. This global representation ensures a consistent input–output structure across training examples. A neural network is then trained to predict the characteristics of all mesh sources – such as location, spacing, and radius of influence, based on geometric and flow parameters. Once trained, the network can be used to infer spacing functions for unseen configurations, supporting rapid mesh generation without requiring a prior solution.

To accelerate the mesh generation stage, a source merging algorithm based on total least-squares is also developed, allowing point sources to be combined into line sources. This significantly reduces the computational overhead of evaluating spacing functions.

Lastly, the proposed methodology is demonstrated through a series of three-dimensional inviscid compressible flow cases, including variations in geometry and flow conditions. The performance of the predicted near-optimal meshes is evaluated by comparing the predicted spacing fields against target spacing, and by assessing the accuracy of the resulting CFD solutions. The environmental and computational impact of the method is also quantified and compared to traditional industrial meshing practices.

5.1 Construction of near-optimal meshes

To obtain an optimum engineering design, numerous geometric configurations must be analysed through the range of operating conditions. Reducing the time to generate a solution with the required accuracy enables more configurations to be evaluated. In addition, often, multiple meshes are required to ensure that the asymptotic convergence has been reached. This process must be repeated for every configuration and for every operating condition. Alternatively, a fine mesh capable of capturing the features for a range of operating conditions can be used. The first process could either be automated – by using mesh adaptivity, or manually controlled – by enhancing and updating the parameters of the used mesh control technique.

This work proposes using historic data, accumulated from analysis carried out during previous designs, to predict an appropriate starting mesh that can be considered a near-optimal for a given geometric configuration and/or operating conditions. It is assumed that solutions satisfying the desired accuracy for a range of geometric configurations and operating conditions are available. These solutions could have been achieved by utilising different modelling techniques, such as structured, unstructured or hybrid methods.

The proposed concept to generate a near-optimal mesh can be summarised in the following four stages:

1. For every solution obtained from a given set of design parameters, create a set of point sources that can generate a mesh to capture the given solution.
2. Combine the individual sets of sources into a global set that is customised to generate the available mesh for all given cases.
3. Train a NN to predict the characteristics of the global set of sources for a new, unseen, set of input parameters.
4. Reduce the predicted global sources and combine point sources into line sources prior to generating the near-optimal mesh.

These four stages are described in detail in the remainder of this Section.

5.1.1 Generating point sources from a given solution

The process of creating a set of sources that leads to a mesh capable of capturing a given solution requires the computation of the suitable element size at every point in space. Here, the element spacing is related to the derivatives of the solution using the Hessian matrix, a principle that is commonly used in error analysis, such as

$$\delta_\beta^2 \left(\sum_{i,j=1}^{N} H_{ij} \beta_i \beta_j \right) = K, \tag{5.1}$$

where $\boldsymbol{\beta}$ is an arbitrary unit vector, δ_β is the spacing along the direction of $\boldsymbol{\beta}$, H_{ij} are the components of the Hessian matrix of a selected key variable σ, namely

$$H_{ij} = \frac{\partial^2 \sigma}{\partial x_i \partial x_j}, \tag{5.2}$$

and K is a user-defined constant. The derivation of Equation (5.1) from fundamental interpolation error analysis is presented in Appendix A, along with a summary of key concepts in error analysis. This background provides the theoretical foundation for relating the local solution error to element sizing.

The derivatives of the key variable, σ, are computed at each node of the current mesh using a recovery process based on a variational residual statement (Zienkiewicz and Zhu, 1992a,b). From this, the Hessian matrix $\mathbf{H}$ is assembled, and its eigenvalues are used to determine the required local mesh spacing. The optimal spacing is computed using the largest eigenvalue, λ, of the Hessian at a given node and is defined as

$$\delta = \begin{cases} \delta_{\text{min}} & \text{if } \lambda > K/\delta_{\text{min}}^2, \\ \delta_{\text{max}} & \text{if } \lambda < K/\delta_{\text{max}}^2, \\ \sqrt{K/\lambda}, & \text{otherwise,} \end{cases} \tag{5.3}$$

where K is a user-defined scaling constant, and δ_{min} and δ_{max} are the minimum and maximum allowable spacing values, respectively.

The spatial distribution of the mesh spacing is determined by the specification of K. In this work, it is taken as $K = \delta_{\text{min}}^2 S^2 \lambda_{\text{max}}$, where S is a scaling factor set to 0.2, and λ_{max} is the maximum eigenvalue of the Hessian over the entire domain.

The maximum spacing value, δ_{max}, is introduced to account for the possibility of very small eigenvalues in Equation (5.3), which could otherwise result in excessively large element sizes. In practice, δ_{max} is used in regions where the solution is smooth and fine resolution is unnecessary. Conversely, large second derivatives typically arise near regions with steep gradients, such as shocks, where smaller elements are required to capture the flow features accurately. A lower bound δ_{min} is imposed to prevent over-refinement in these regions, limiting the minimum allowable element size and avoiding excessive concentration of mesh elements.

To enable the use of data generated using different numerical techniques, a method that ensures the uniformity of the training data has to be devised. Since the underlying meshes that were used for generating the data can be different and are often very large, the use of sources to specify the mesh requirement to capture the given solutions is proposed.

The main idea is, for a given mesh, to group points that have similar required spacing, calculated using Equation (5.3), to form a point source that is located at the centre of the grouped points with a radius that extends from the centre to the furthest point in the group. This process reduces the quantity of information needed to describe an optimal mesh by up to two orders of magnitude without sacrificing the quality of the information which describes the required mesh.

It is worth noting that the process will reflect the level of fidelity provided by the given solutions. Hence, the process can be used both at the initial investigation stage – when the meshes are not highly refined, and at the data acquisition stage, when the solutions are provided on a highly refined mesh or on an adapted mesh.

Algorithm 1 describes the devised technique to convert a given solution to a set of point sources. The process starts from the computed spacing required at every point of the mesh to capture a given solution. To ensure optimum grouping of points that have similar required spacing, the number of surrounding layers of nodes with spacing similar to the node under consideration is computed. Figure 5.1 illustrates the concept of surrounding layers.

The process ensures that a minimum of one point source covers the spacing required at every node of the given mesh. In the current implementation, it is assumed that the radius R, where the spacing doubles, is twice the radius r where the spacing is constant and equal to the spacing of the grouped points. It is also assumed that different spacings are considered *similar*, if they are lower than 5% of the spacing at the node under consideration. The creation of a point source is terminated if the spacing at a surrounding layer is larger than the spacing at the radius R of the point source.

Algorithm 1 Process for creating point sources

Input: A mesh $\mathcal{M}$ with N nodes and a vector $\boldsymbol{\delta}$ of dimension N, where the component δ_I contains the spacing at node I calculated using the equidistribution principle, as given by Equation (5.3). The set of nodes is denoted by $\mathcal{N} = \{1, \ldots, N\}$ and the coordinates of the I-th node are denoted by $\boldsymbol{x}_I$

```
 1: Construct a vector $\boldsymbol{l}$ of dimension $N$, where the component $l_I$ contains the number
    of surrounding layers of nodes with a spacing $\delta_J$ such that $\delta_J/\delta_I < \gamma$, with $\gamma$ being
    a user-specified ratio;
 2: Initialise the set of unused nodes: $\mathcal{U} \leftarrow \mathcal{N}$;
 3: Initialise the set of sources: $\mathcal{S} \leftarrow \emptyset$;
 4: while $|\mathcal{U}| > 0$ do
 5:     Find $J \in \mathcal{U}$ such that $\delta_J < \delta_K$ and $l_J > l_K, \ \forall K \in \mathcal{U}$;
 6:     Initialise a new point source: $\lambda \leftarrow \{\boldsymbol{x}_J, \delta_J, r = \delta_J, R = 2r\}$;
 7:     Create a list: $\mathcal{L} \leftarrow \{J\}$;
 8:     Mark the vertex $J$ as used: $\mathcal{U} \leftarrow \mathcal{U} \setminus \{J\}$;
 9:     $N_{\mathcal{L}} \leftarrow 0$;
10:     while $N_{\mathcal{L}} \neq |\mathcal{L}|$ do
11:         $N_{\mathcal{L}} \leftarrow |\mathcal{L}|$;
12:         Create a list: $\tilde{\mathcal{L}} \leftarrow \{K \in \mathcal{N} \mid K$ is connected to a node in $\mathcal{L}\}$;
13:         if $\delta_K/\delta_J < \gamma, \ \forall K \in \tilde{\mathcal{L}}$ then
14:             $\mathcal{L} \leftarrow \mathcal{L} \cup \tilde{\mathcal{L}}$;
15:             $\mathcal{U} \leftarrow \mathcal{U} \setminus \tilde{\mathcal{L}}$;
16:             Modify source: $\lambda \leftarrow \left\{ \sum_{S \in \mathcal{L}} \boldsymbol{x}_S/|\mathcal{L}|, \delta_J, r = \max_{S \in \mathcal{L}}\{\|\boldsymbol{x}_{\mathcal{L}} - \boldsymbol{x}_S\|_2\}, R = 2r \right\}$;
17:         else if $\exists K \in \tilde{\mathcal{L}} \mid \delta_K/\delta_J < 2$ then
18:             for $K \leftarrow 1$ to $|\tilde{\mathcal{L}}|$ do
19:                 if $\delta_K/\delta_J < \gamma$ then
20:                     $\mathcal{L} \leftarrow \mathcal{L} \cup \{K\}$
21:                     $\mathcal{U} \leftarrow \mathcal{U} \setminus \{K\}$
22:                     Modify source: $\lambda \leftarrow \{\boldsymbol{x}_J, \delta_J, r = \|\boldsymbol{x}_I - \boldsymbol{x}_K\|_2, R = 2r\}$;
23:                 else if $\delta_K/\delta_J < 2$ then
24:                     $\mathcal{L} \leftarrow \mathcal{L} \cup \{K\}$;
25:                     $\mathcal{U} \leftarrow \mathcal{U} \setminus \{K\}$
26:                 end if
27:             end for
28:         end if
29:     end while
30:     $\mathcal{S} \leftarrow \mathcal{S} \cup \{\lambda\}$;
31: end while
```

Output: A set of point sources $\mathcal{S} = \{\lambda\}$, each defined by its centre $\boldsymbol{x}_J$, spacing δ_J, and radii r and R.

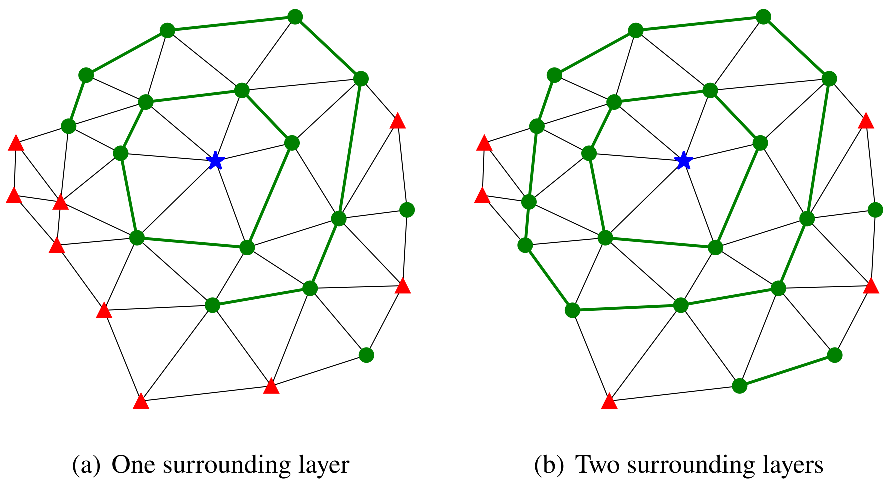


Figure 5.1: Detail of a triangular mesh illustrating the concept of surrounding layers. The central node is denoted by a blue star. The nodes with similar spacing are denoted with a green circle, whereas nodes with a dissimilar spacing are denoted with a red triangle. The scenario in (a) shows one surrounding layer of nodes with similar spacing to the central node, whereas the scenario in (b) shows two surrounding layers.

5.1.2 Generating global sources from sets of local sources

The process described above does not guarantee that the number of sources to capture the required meshes is the same for different sets of input parameters. In addition, despite the fact that two sets of sources would contain sources in close proximity, their position in the list of sources can be, in general, very different. The former will make the use of NNs unfeasible, whereas the latter will impose significant difficulties when finding a correlation between inputs and outputs. To solve these two issues, a process to create one set of global sources that can be used for all sets of input parameters is devised.

The basic idea is to create one set of global sources that can be used to map the index of each source in a local set to an index of a source in the global set. The mapping is based on the minimum distance between the global and local sources, and it is defined as

$$\begin{aligned} \mathcal{F} \ &: L \longrightarrow G \\ (j,i) &\longmapsto \mathcal{F}(j,i) := \begin{cases} k & \text{if } \exists k \in G \,|\lambda_{j,i} = \Lambda_k \\ 0 & \text{otherwise} \end{cases}, \end{aligned} \tag{5.4}$$

where $L_i = \{(j,i) \mid j \in \{1,\ldots,|\mathcal{S}_i|\}\}$ denotes the indices of local sources in $\mathcal{S}_i$, the indices of all local sources is $L = L_1 \cup \ldots \cup L_C$ and the indices of the global sources is $G = \{1,\ldots,|\mathcal{G}|\}$, where $\mathcal{G}$ is the list of global sources.

Algorithm 2 describes the developed process to combine the sets of local sources into one set of global sources. The set of global sources are further tuned for each input set of parameters to produce the required mesh to capture the given solution.

The process starts with an empty list of global sources and considers one case at a time. As the sources of the first available case are unique, they are added to the list of global sources with a one-to-one mapping. Any newly added source to the global list is inserted in an ADT data structure. The ADT is then utilised to identify sources from the global list that are in close proximity to the remaining local sources. If no source from the global list is close to a local source, the local source is added to the list of global sources and the mapping is updated. If a global source is found to be in close proximity to a local source, the local source index is mapped to the global source index. When all cases are considered, the global set of sources are customised for each set of input parameters. For each case, if a local source has been mapped to a global source, the local source characteristics are used unchanged. For any global source that has no corresponding local source, the characteristics are calculated to ensure that the spacing produced is compatible with the spacing that the local source would produce.

Figure 5.2 illustrates the two algorithms described above using a two-dimensional example with two design variables: the free-stream Mach number and the angle of attack. For each training case, the pressure field is used to obtain a set of point sources. The point sources in Figure 5.2 are represented by circles, coloured using the spacing value and with a radius proportional to the radius of influence of each source. The

Algorithm 2 Process for creating global sources

Input: Set of lists of point sources $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_C\}$, where $\mathcal{S}_i$ is the list of sources for the i-th case and C is the total number of cases. The j-th source of the i-th case is denoted by $\lambda_{j,i}$ and the sphere associated to a point source, with centre $\boldsymbol{x}_{j,i}$ and radius $r_{j,i}$, is denoted by $\mathcal{B}_{j,i} = \{\boldsymbol{x} \in \mathbb{R}^3 \mid \|\boldsymbol{x} - \boldsymbol{x}_{j,i}\|_2 \leq r_{j,i}\}$;

```
 1: Insert the sources of all cases, $\mathcal{S}$, in an alternating digital tree (ADT);
 2: Initialise the list of global sources: $\mathcal{G} \leftarrow \mathcal{S}_1$. Global sources are denoted by $\Lambda_l$,
    for $l = 1, \ldots, |\mathcal{G}|$ and the sphere associated to a global source with centre $\boldsymbol{x}_l$ and
    radius $r_l$, is denoted by $\mathcal{B}_l = \{\boldsymbol{x} \in \mathbb{R}^3 \mid \|\boldsymbol{x} - \boldsymbol{x}_l\|_2 \leq r_l\}$;
 3: Associate sources in $\mathcal{S}_1$ with corresponding global sources in $\mathcal{G}$ by using the
    mapping given by Equation (5.4);
 4: for $i \leftarrow 2$ to $C$ do
 5:     for $j \leftarrow 1$ to $|\mathcal{S}_i|$ do
 6:         Search the ADT to create a list of sources: $\mathcal{L} \leftarrow \{\lambda_{r,s} \in \mathcal{S} \mid \mathcal{B}_{r,s} \cap \mathcal{B}_{j,i} \neq \emptyset\}$;
 7:         for all $\lambda_{r,s} \in \mathcal{L}$ do
 8:             if $F(r,s) \neq 0$ and $\|\boldsymbol{x}_{j,i} - \boldsymbol{x}_{r,s}\|_2 \leq \min\{r_{j,i}, r_{r,s}\}$ then
 9:                 Find $\lambda_{u,v}$ such that $\|\boldsymbol{x}_{u,v} - \boldsymbol{x}_{j,i}\|_2 \leq \min_{\lambda_{a,b} \in \mathcal{L}} \{\|\boldsymbol{x}_{a,b} - \boldsymbol{x}_{j,i}\|_2\}$;
10:             end if
11:         end for
12:         if $\exists\ \lambda_{u,v}$ then
13:             Associate $\lambda_{j,i}$ with the global source $\Lambda_{F(u,v)}$;
14:         else
15:             $\mathcal{G} \leftarrow \mathcal{G} \cup \{\lambda_{j,i}\}$;
16:             Associate $\lambda_{j,i}$ with the global source $\Lambda_{|\mathcal{G}|}$;
17:         end if
18:     end for
19: end for
20: for $i \leftarrow 1$ to $C$ do
21:     Initialise the list of combined sources: $\mathcal{C}_i \leftarrow \emptyset$;
22:     for $k \leftarrow 1$ to $|\mathcal{G}|$ do
23:         if $\exists j \mid \mathcal{F}(j,i) = k$ then
24:             $\lambda^\star \leftarrow \lambda_{j,i}$;
25:         else
26:             $\mathcal{A} \leftarrow \{(r,s) \mid \mathcal{F}(r,s) = k\}$;
27:             $\lambda^\star \leftarrow \left\{ \sum_{(r,s) \in \mathcal{A}} \boldsymbol{x}_{r,s} / \|\mathcal{A}\|, \delta^\star = \min_{l \in \mathcal{S}_i} \{\delta_l(\boldsymbol{x}^\star)\}, r = \delta^\star, R = 2r \right\}$;
28:         end if
29:         $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{\lambda^\star\}$
30:     end for
31: end for
```

Output: Set of lists of global sources $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_C\}$, where $\mathcal{C}_i$ is the list of sources for the i-th case, containing $|G|$ sources.

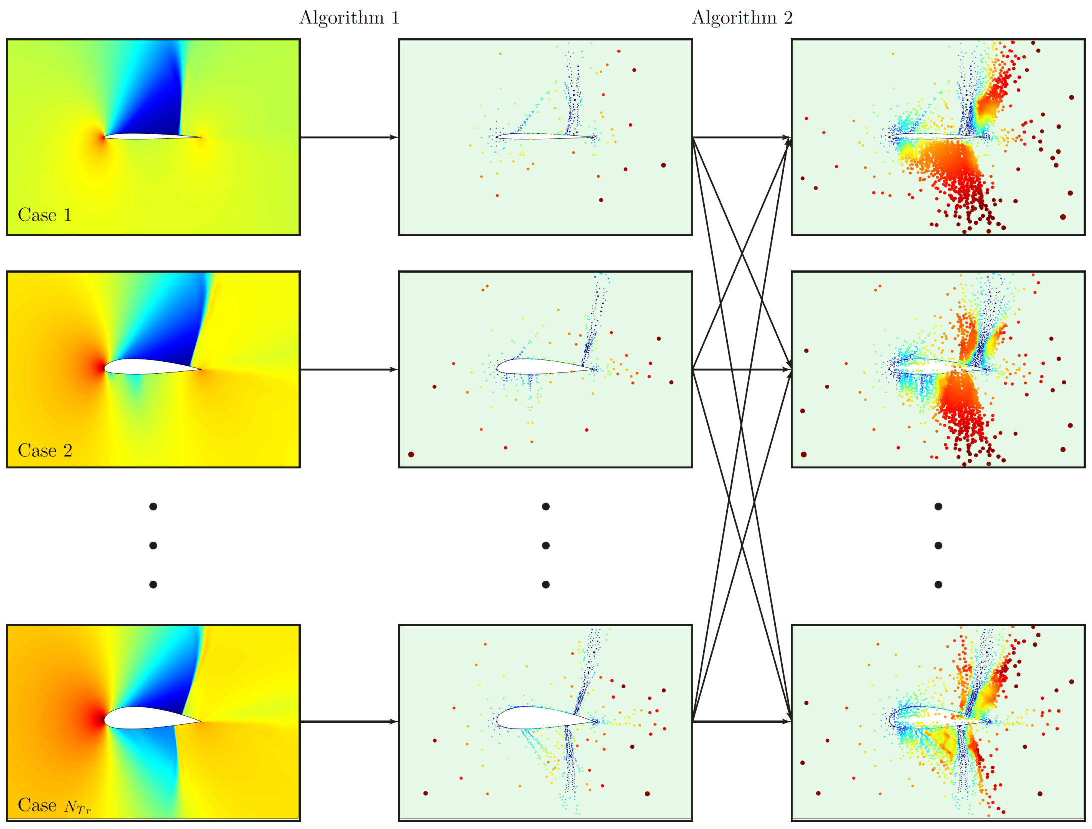


Figure 5.2: Illustration of the process described in Algorithms 1 and 2 to obtain the point sources for a set of training cases and the construction of the global set of sources.

illustration, made with real data from a two-dimensional example, shows that point sources with small spacing are created in the regions where the solution shows a high pressure gradient.

Once the point sources for all cases are constructed, the sets of global sources are created for each case using Algorithm 2. This process ensures that the number of global sources is the same in all cases, and therefore, the data can be used to train a NN.

5.1.3 Construction of NN for predicting the characteristics of the sources

The creation of a set of global sources with the same number of characteristics for all cases enables the use of NNs to predict these characteristics for unseen cases. In this work, the objective is to predict the location, spacing and the radius within which the spacing remains constant. Generally, the values of the spacing and the radius varies by more than two orders of magnitude. The logarithm of the spacing and the radius is used to ensure consistent training of the NN, without a bias towards larger values. This scaling also prevents the NN from predicting unphysical negative values for these two outputs.

As stated in Section 4.2.4, the NN has an input vector $\mathbf{x} = \{x_1, ..., x_N\}^T$, and an output vector $\mathbf{y} = \{y_1, ..., y_M\}^T$. In the numerical examples considered in this work, the number of inputs, N, is related to the flow conditions and/or geometric parameters, whereas the number of outputs M are the source characteristics (i.e., position, spacing and radius r). When N_{tr} training cases are considered, the input is an array of size $N \times N_{tr}$ and the output is an array of size $M \times N_{src} \times N_{tr}$, where N_{src} is the number of global sources.

Different models can be used to train and predict the characteristics of the global sources. The obvious choice is to train one NN to predict all the source characteristics. However, this work will also investigate the use of different NNs to train source characteristics of different nature. For instance, one NN can be trained to predict the position of the sources, another NN to predict the spacing and another NN to predict the radius. Another alternative would be to train N_{src} NNs, each one trained to predict the characteristics of a single source.

In terms of the implementation, TensorFlow 2.7.0 (Abadi et al., 2016) was used to construct the NNs considered in this work. To ensure that the NN prediction capability is not heavily influenced by the initial choice of the NN weights, the training is performed five times for each experiment considered by varying the seed values of the optimisation process. For each training, a maximum of 500 epochs is considered, and the training is stopped when either the maximum number of epochs is reached or no improvement in the objective function is observed during 50 consecutive epochs. It is worth noting that the batch size used in all the examples considered is eight, which for the examples presented here produced a better performance than the default value of 32 in TensorFlow.

The selection of an appropriate optimiser and activation function is critical to the stability and accuracy of neural network training. A detailed study of various optimiser and activation function combinations is presented in Appendix C. Based on the findings of that study, the combination of the Adam optimiser (see Section 4.3.2) with the Sigmoid activation function (defined in Section 4.2.2) is adopted for all NN training throughout this thesis. This configuration was found to offer the best trade-off between accuracy, robustness, and efficiency. It consistently delivers strong predictive performance without requiring extensive hyperparameter tuning, particularly when supported by a broad sampling of the network architecture.

The neural network design in this work involves selecting suitable values for key hyperparameters, including the number of layers, neurons per layer, and activation functions. For each numerical example in Section 5.2, a grid search is performed over the ranges $N_l = [1, 2, \dots, 6]$ and $N_n = [25, 50, \dots, 250]$ to identify an optimal configuration.

To measure the accuracy of the NN predictions, the classical statistical R^2 measure (Glantz and Slinker, 2001) is considered for all the test cases. To better illustrate the difficulty of predicting the different characteristics of mesh sources, the R^2 measure is reported for the five characteristics independently (x, y and z position of the source, spacing and radius).

5.1.4 Reducing the global set of sources

After the training stage, a NN is used to predict the characteristics of the global set of sources for unseen cases. Despite the fact that it is possible to use the global set of sources to generate a predicted mesh, removing from the list of global sources entries that are redundant has the potential to speed up the generation process considerably. This is because, during the mesh generation process, it is necessary to compute the spacing induced by each source at a point in order to take the minimum of all the spacings.

Sources with an associated spacing function adequately described by other sources are classed as redundant. Algorithm 3 describes the process devised to remove redundant sources. The algorithm starts by determining the maximum region of influence of

Algorithm 3 Process for removing inactive point sources

```
    Input: List of global point sources $\mathcal{G} = \{\Lambda_1, \dots, \Lambda_{|\mathcal{G}|}\}$;
 1: Create a vector, $\boldsymbol{d}$, of dimension $|\mathcal{G}|$, where the $i$-th component is $d_i$ such that $\delta(\boldsymbol{x}) = \tilde{\delta}, \forall \boldsymbol{x} \in \partial \mathcal{B}_{d_i, \boldsymbol{x}_i}$, with $\tilde{\delta}$ being the background spacing and $\mathcal{B}_{d, \boldsymbol{x}_0} = \{\boldsymbol{x} \in \mathbb{R}^3 \mid \|\boldsymbol{x} - \boldsymbol{x}_0\|_2 \leq d\}$;
 2: Insert the sources in an ADT;
 3: Mark all sources as active: $\mathcal{A} \leftarrow \mathcal{G}$
 4: for $i \leftarrow 1$ to $|\mathcal{G}|$ do
 5:     Search the ADT to create a list of sources: $\mathcal{L} \leftarrow \{\Lambda_k \in \mathcal{G} \mid \mathcal{B}_k \cap \mathcal{B}_i \neq \emptyset\}$;
 6:     for $j \leftarrow 1$ to $|\mathcal{L}|$ do
 7:         if $\Lambda_j \in \mathcal{A}$ then
 8:             if $\delta_i < \delta_j$ and $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 < r_i - r_j$ then
 9:                 $\mathcal{A} \leftarrow \mathcal{A} \setminus \{\Lambda_j\}$;
10:             else if $\delta_i > \delta_j$ and $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 < r_j - r_i$ then
11:                 $\mathcal{A} \leftarrow \mathcal{A} \setminus \{\Lambda_i\}$;
12:                 break;
13:             end if
14:         end if
15:     end for
16: end for
17: Initialise the list of visited sources: $\mathcal{V} \leftarrow \emptyset$;
18: Initialise the list of reduced sources: $\mathcal{R} \leftarrow \emptyset$;
19: while $|\mathcal{V}| < |\mathcal{A}|$ do
20:     Find $\Lambda_k \in \mathcal{A}$ such that $\delta_k = \min_{\Lambda_j \in \mathcal{A}} \{\delta_j\}$ and $r_k = \max_{\Lambda_j \in \mathcal{A}} \{r_j\}$;
21:     Define a Cartesian grid of $\mathcal{H} := [x_k - R_k, x_k + R_k] \times [y_k - R_k, y_k + R_k] \times [z_k - R_k, z_k + R_k]$ with $\lceil 2R_k/\delta_k \rceil$ equally-spaced points in each direction and define the sampling points in the grid as $\boldsymbol{p}_{r,s,t}$
22:     if $\exists \boldsymbol{p}_{r,s,t} \mid \delta_k(\boldsymbol{p}_{r,s,t}) < \min_{\Lambda_j \in \mathcal{R}} \{\delta_j(\boldsymbol{p}_{r,s,t})\}$ then
23:         $\mathcal{R} \leftarrow \mathcal{R} \cup \{\Lambda_k\}$
24:     end if
25:     $\mathcal{V} \leftarrow \mathcal{V} \cup \{\Lambda_k\}$;;
26: end while
    Output: A reduced set of global sources $\mathcal{R} = \{\lambda_1, \dots, \lambda_{|\mathcal{R}|}\} \subseteq \mathcal{G}$, containing only the active and non-redundant sources required to represent the spacing field.
```

each source, i.e. the region where the computed spacing from the source reaches the maximum allowable spacing. All sources are inserted into an ADT data structure to

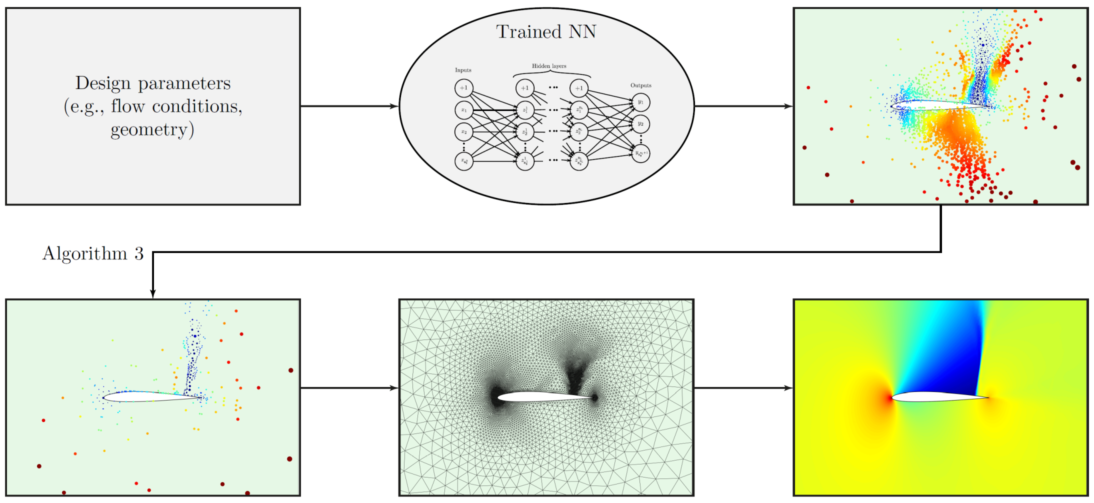


Figure 5.3: Illustration of prediction stage using the trained NN and the process described in Algorithms 3 to reduce the set of sources before generating and mesh that can finally be used to compute a solution.

speed up the search process. For each source in the global list, the ADT is searched to identify sources that have an overlapping sphere with the source under consideration. The source with a larger spacing is removed if its associated sphere is inside the associated sphere of the source with smaller spacing.

A further check is conducted to determine if the region of influence of a source can be covered by multiple regions of influence of other sources. This is performed in a discrete fashion by using the spacing of the source to divide the region of influence of the source under consideration into a uniform local grid. The spacing at each point of the local grid is evaluated using the sources that have been already added to the list of reduced sources. If the evaluated spacing at any point of the local grid has a larger spacing than the spacing from the source under consideration, the source is added to the list of reduced sources.

Figure 5.3 illustrates the on-line stage of the proposed approach. After the NN is trained for a new set of design parameters (flow conditions in this two-dimensional example), the NN is used to predict the characteristics of the global sources. Using Algorithm 3, the sources are reduced. This process ensures that the mesh generation stage does not require a very large number of queries to compute the spacing at a point. For the examples considered in this work, Algorithm 3 produces a reduction of around 60% in the number of point sources.

Once the mesh is obtained, the standard CFD calculation is performed. The illustrative example of Figure 5.3 shows the computed pressure field, which exhibits all the expected flow features for this transonic case.

In an attempt to produce a smooth spacing function, line sources are crated using group point sources. Algorithm 4 describes the process that is proposed for the creation of line sources from grouped point sources. An iterative process is used to identify sources that have similar spacing, and their associated sphere intersects with the associated sphere of a source that has already been added to the group. Orthogonal regression is then used to fit a plane to the grouped point sources. The distance of each point source to the plane is calculated, and the points with a distance greater than their radius of influence are removed from the group. The process of finding a best fit plane continues until all points in the group are within the allowable distance. The coordinates of the remaining point sources are projected onto the best fit plane and used to compute the best fit line using orthogonal regression. The process of removing points from the group that do not satisfy the allowable distance criteria will also be applied to the projected points. The furthest two points remaining in the group will be used to form a line source.

Algorithm 4 Process for merging point sources into line sources

```
    Input: List of point sources: $\mathcal{L} = \{\Lambda_1, \dots, \Lambda_{|\mathcal{L}|}\}$;
 1: Initialise the set of unused sources: $\mathcal{U} \leftarrow \mathcal{L}$;
 2: while $\mathcal{U} \neq \emptyset$ do
 3:     Initialise the list of connected point sources: $\mathcal{C} \leftarrow \emptyset$;
 4:     Find $\Lambda_k \in \mathcal{U}$ such that $\delta_k = \min_{\Lambda_j \in \mathcal{L}} \{\delta_j\}$;
 5:     Create the list of sources to be added: $\mathcal{A} \leftarrow \{\Lambda_k\}$;
 6:     Add sources to the list: $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{A}$;
 7:     Mark sources as used: $\mathcal{U} \leftarrow \mathcal{U} \setminus \mathcal{A}$;
 8:     $N_{\mathcal{C}} \leftarrow 0$;
 9:     while $N_{\mathcal{C}} \neq |\mathcal{C}|$ do
10:         $N_{\mathcal{C}} \leftarrow |\mathcal{C}|$;
11:         $\tilde{\mathcal{A}} \leftarrow \emptyset$
12:         for $j \leftarrow 1$ to $|\mathcal{A}|$ do
13:             for $i \leftarrow 1$ to $|\mathcal{L}|$ do
14:                 if $\Lambda_i \in \mathcal{U}$ and $\max\{\delta_i, \delta_k\} / \min\{\delta_i, \delta_k\} < \gamma$ and $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 < r_i + r_j$ then
15:                     $\tilde{\mathcal{A}} \leftarrow \tilde{\mathcal{A}} \cup \{\Lambda_i\}$;
16:                 end if
17:             end for
18:         end for
19:         $\mathcal{A} \leftarrow \tilde{\mathcal{A}}$;
20:         $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{A}$;
21:         $\mathcal{U} \leftarrow \mathcal{U} \setminus \mathcal{A}$;
22:     end while
23:     $N_{\mathcal{C}} \leftarrow 0$;
24:     while $N_{\mathcal{C}} \neq |\mathcal{C}|$ do
25:         $N_{\mathcal{C}} \leftarrow \|\mathcal{C}\|$;
26:         $\mathcal{X} \leftarrow \{\boldsymbol{x}_i\}_{i=1,\dots,|\mathcal{C}|}$;
27:         Compute the plane $\mathbb{P}$ that fits the set of points $\mathcal{X}$ using orthogonal regression, as described in Appendix B.1;
28:         for $i = 1$ to $|\mathcal{C}|$ do
29:             Compute the distance to the plane: $d_{i,\mathbb{P}} := \min_{\boldsymbol{x} \in \mathbb{P}} \{\|\boldsymbol{x} - \boldsymbol{x}_i\|_2\}$;
30:             if $d_{i,\mathbb{P}} > r_i$ then
31:                 $\mathcal{C} \leftarrow \mathcal{C} \setminus \{\Lambda_i\}$;
32:                 $\mathcal{U} \leftarrow \mathcal{U} \cup \{\Lambda_i\}$;
33:             end if
34:         end for
35:     end while
36:     $N_{\mathcal{C}} \leftarrow 0$
37:     while $N_{\mathcal{C}} \neq |\mathcal{C}|$ do
38:         $N_{\mathcal{C}} \leftarrow |\mathcal{C}|$;
39:         $\mathcal{X} \leftarrow \{\boldsymbol{x}_i\}_{i=1,\dots,|\mathcal{C}|}$;
40:         Compute the line $\mathbb{L}$ that fits the set of points $\mathcal{X}$ using orthogonal regression, as described in Appendix B.2;
```

```
41:         for $i = 1$ to $|\mathcal{C}|$ do
42:             Compute the distance to the plane: $d_{i,\mathbb{L}} := \min_{\boldsymbol{x} \in \mathbb{L}}\{\|\boldsymbol{x} - \boldsymbol{x}_i\|_2\}$;
43:             if $d_{i,\mathbb{L}} > r_i$ then
44:                 $\mathcal{C} \leftarrow \mathcal{C} \setminus \{\Lambda_i\}$;
45:                 $\mathcal{U} \leftarrow \mathcal{U} \cup \{\Lambda_i\}$;
46:             end if
47:         end for
48:     end while
49:     Find $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ such that $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 = \max_{\Lambda_k, \Lambda_l \in \mathcal{C}} \|\boldsymbol{x}_k - \boldsymbol{x}_l\|_2$;
50:     Build a line source using the point sources $\Lambda_i$ and $\Lambda_j$;
51: end while
```

Output: A set of line sources $\mathcal{L} = \{\ell_1, \dots, \ell_{|\mathcal{L}|}\}$, where each line source ℓ_k is defined by two endpoints Λ_i and Λ_j.

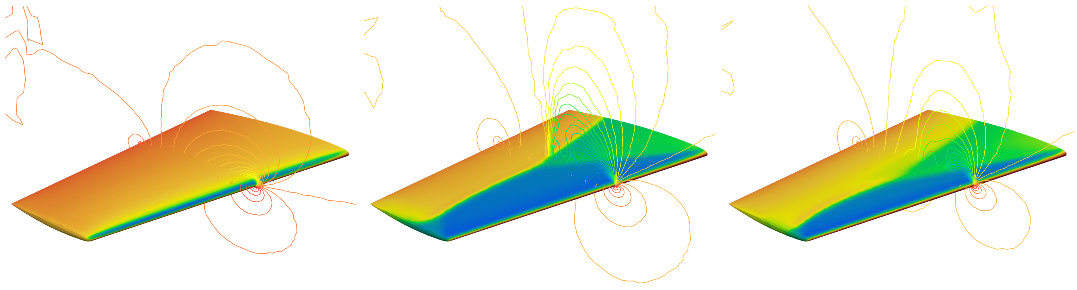
5.2 Numerical examples

This Section presents three numerical examples of increasing difficulty to test the potential and applicability of the proposed technique. The examples involve the prediction of meshes for three-dimensional CFD simulations involving inflow conditions and geometric parameters. One of the challenges of the examples used is that the variation of the input parameters induces different flow patterns that include subsonic and transonic cases with different shock structures and strengths. Therefore, being able to predict the near-optimal mesh for unseen cases is not an easy task.

5.2.1 Near-optimal mesh predictions on the ONERA M6 wing at various inflow conditions

The first example considers the prediction of near-optimal meshes over a fixed geometry, with variable flow conditions. The objective is to study the potential of the proposed approach to accurately predict the sources that can be used to generate near-optimal meshes for unseen flow conditions. Numerical experiments are used to study the influence of the hyperparameters and the size of the training set.

The geometry used in this example is the ONERA M6 wing (Schmitt, 1979) and the



(a) $M_\infty = 0.41, \alpha = 8.90°$ (b) $M_\infty = 0.80, \alpha = 8.19°$ (c) $M_\infty = 0.79, \alpha = 5.39°$

Figure 5.4: Pressure coefficient, C_p, for three different flow conditions used as reference cases throughout this section. (a) A subsonic case, (b) a randomly selected transonic case, and (c) the worst-performing transonic case in terms of mesh spacing prediction accuracy.

inviscid compressible flow conditions are described by two parameters, namely the free-stream Mach number, M_∞, and the angle of attack, α. It is worth noting that the range used for the parameters, $M_\infty \in [0.3, 0.9]$ and $\alpha \in [0°, 12°]$ leads to subsonic and transonic flows. This means that the required meshes vary substantially with respect to the flow conditions.

To illustrate the variation in the solution induced by the parameters, Figure 5.4 shows the pressure coefficient, C_p, for three representative combinations of the input parameters. These three cases are used throughout this section as recurring examples to evaluate the mesh quality and flow solution fidelity.

For the first set of input parameters, $M_\infty = 0.41$ and $\alpha = 8.90°$, the flow is subsonic, and the mesh should be refined near the leading and trailing edges to capture the high variation of the pressure on these regions. For the second set of inputs, $M_\infty = 0.80$ and $\alpha = 8.19°$, the typical λ-shock can be clearly observed. Refinement near the discontinuity of the pressure is, therefore, necessary to capture such abrupt changes in the pressure field. Finally, for the last set of inputs, $M_\infty = 0.79$ and $\alpha = 5.39°$, the λ-shock is also clearly visible, but at a different position and with a different strength when compared to the second case. The simulations were performed using the in-house flow solver FLITE (Sørensen et al., 2003a) with unstructured tetrahedral meshes consisting of approximately 1.3M elements and 230K nodes. The surface mesh is made of approximately 20K triangular elements.

These three cases were selected to highlight different flow phenomena and to assess the robustness of the proposed method across a range of conditions. The first case corresponds to a simple subsonic regime with smooth flow. The second case represents a randomly selected transonic condition, where a λ-shock is clearly present. Lastly, the third case is the most challenging among the test cases, exhibiting the largest spacing error in the mesh prediction. By including this worst-performing case, the evaluation seeks to demonstrate not only the overall accuracy of the method but also its performance under difficult conditions.

This structured selection process is applied throughout the remainder of the thesis, with random cases and the latter being the most difficult case within the test set. In all examples, the aim is to present a representative range of conditions while ensuring that any limitations of the method are directly examined.

For training, a set with $N_{tr} = 160$ cases is considered, and a set with $N_{tst} = 100$ cases is considered for testing. To minimise the undesired use of the trained NNs for extrapolation, the training set is generated in the region of interest, namely $(M_\infty, \alpha) \in [0.3, 0.9] \times [0^\circ, 12^\circ]$, whereas the test set is generated using a reduced space, namely $(M_\infty, \alpha) \in [0.33, 0.81] \times [1.0^\circ, 11.0^\circ]$. Both the training and testing datasets are displayed in Figure 5.5.

To produce sampling points that offer good coverage of the parametric space, the training and testing datasets are generated using scrambled Halton sequencing. The Halton sequence is a sequence of points commonly used in numerical analysis and Monte Carlo simulations because it is deterministic, and the points generated have low discrepancy (Cheng and Druzdzel, 2000). The scrambled Halton sequence is a modification of the original Halton sequence to improve performance in higher dimensions (Halton, 1964; Vandewoestyne and Cools, 2006).

It should be noted that when utilising data generated by industry over the years, the

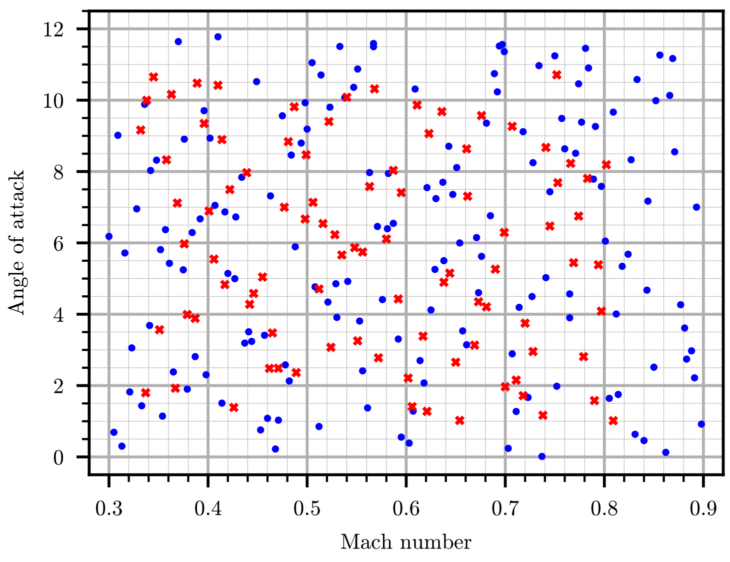


Figure 5.5: Data sets for training (blue circles) and testing (red crosses) for the example with varying flow conditions.

training set will most likely not correspond to a Halton sequencing. In fact, it would be expected that the cases available have been decided by an expert engineer and, therefore, would be specifically selected to capture sensitive areas of the flight envelope that undergo large changes in flow features. Such bias in parameter selection is expected to improve the accuracy of the NNs. As such, the random approach of the Halton sequencing taken in this work is a more conservative approach that requires more training data.

The number of sources generated from the solutions of the training and test datasets, computed by using Algorithm 1, varied between 2,142 and 5,593. The global set of sources that resulted from the combination of sources described in Algorithm 2 and used for training the NNs consisted of 19,345 sources.

Three different models could be used for predicting the characteristics of the sources. All the models use the same two inputs but differ by the number of NNs required and the number of predicted outputs for each NN. The first model predicts the five characteristics of all the sources using a single NN. The second model trains the location (three coordinates) of all the sources using one NN, whereas the spacing and radius are trained using two more NNs. The third model trains each characteristic of

Table 5.1: Three models used for training NNs able to predict the characteristics of the sources.

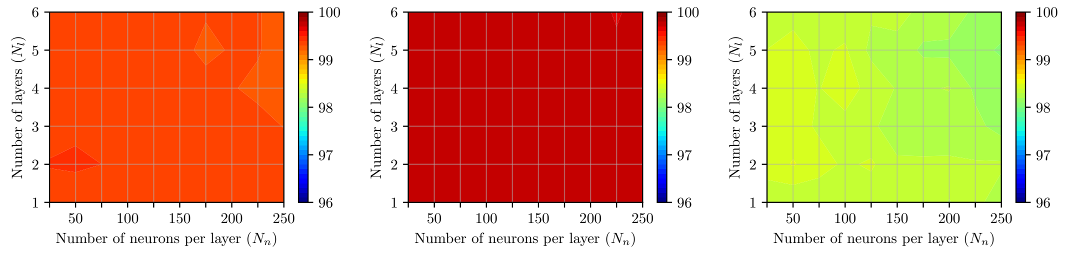
Model	NN architecture				
1	NN$_1$ (x, y, z, δ_0, r)				
2	NN$_1$ (x, y, z)	NN$_2$ (δ_0)	NN$_3$ (r)		
3	NN$_1$ (x)	NN$_2$ (y)	NN$_3$ (z)	NN$_4$ (δ_0)	NN$_5$ (r)

all the sources separately, requiring the training of five NNs. The three models under consideration are summarised in Table 5.1.

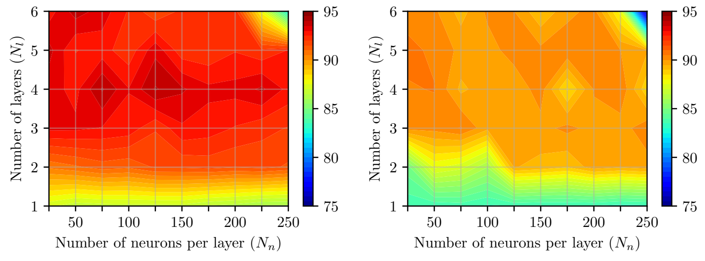
Three further models could be considered, where the training of each source is performed independently. For instance, N_{src} NNs could be trained to predict the five characteristics of each source. Similarly, three NNs could be trained per source or even five NNs per source. These three models are not considered in the current work because they become impractical for realistic problems involving a large number of sources. Not only the training requires the construction of thousands of NNs, but also the prediction stage requires loading the weights of thousands of NNs to compute the required spacing.

To compare the three models described in Table 5.1, first the hyperparameters of each NN are tuned. Figure 5.6 shows the variation of the R^2 as a function of the number of layers and the number of neurons in each layer when using $N_{Tr} = 160$ training cases. It is worth noting that a different colour scale is used for the coordinates and the remaining two characteristics, i.e. spacing and radius. The results show that it is substantially more difficult to predict the spacing and radius, compared to the coordinates of the source. In addition, the accuracy of the predicted coordinates is almost insensitive to the NN architecture, whereas the spacing and radius benefit from using a particular choice of the hyperparameters. The most suitable architecture for this example would be a NN with three hidden layers and 25 neurons per layer. This will provide the best possible accuracy for all the characteristics whilst minimising the size of the NN.

Figure 5.7 shows the regression plot for the spacing, for three flow conditions that

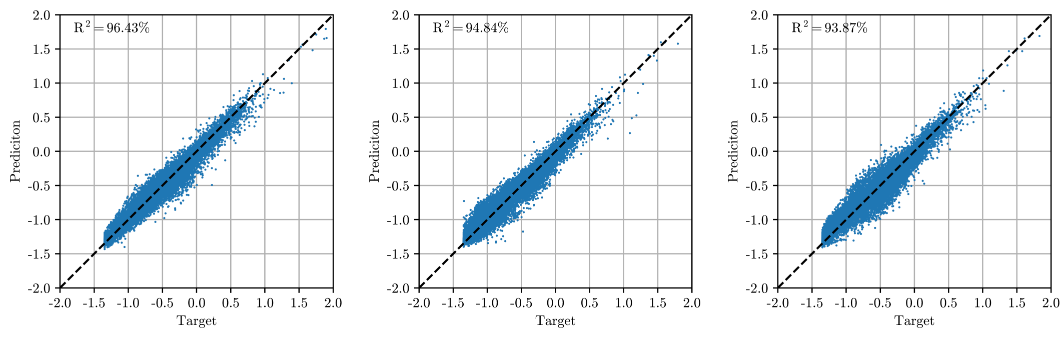


(a) x (b) y (c) z



(d) δ_0 (e) r

Figure 5.6: R^2 for the five source characteristics as a function of the number of layers and number of neurons in each layer for model 2 and for the example with varying flow conditions.



(a) $M_\infty = 0.41, \alpha = 8.90°$ (b) $M_\infty = 0.80, \alpha = 8.19°$ (c) $M_\infty = 0.79, \alpha = 5.39°$

Figure 5.7: The regression plots for the spacing, δ_0, for three flow conditions corresponding to a subsonic and two transonic cases.

correspond to a subsonic and two transonic cases. It is worth noting that the R^2 value for all cases is above 90%, despite the substantial difference in flow features between a subsonic and a transonic case. This shows the robustness of the proposed approach when predicting the spacing of each source, which is the most difficult quantity to predict accurately.

The histogram in Figure 5.8 shows the R^2 for all test cases. The results show that only one test case has an R^2 value below 90, and the average R^2 is above 96. This

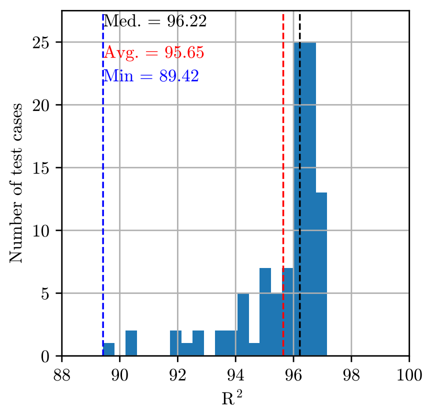
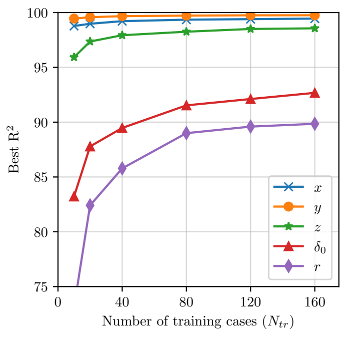
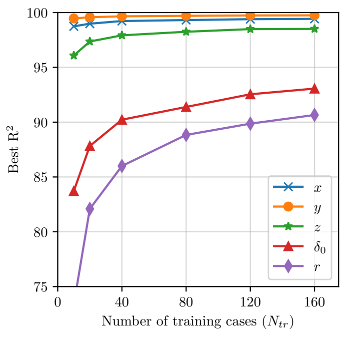


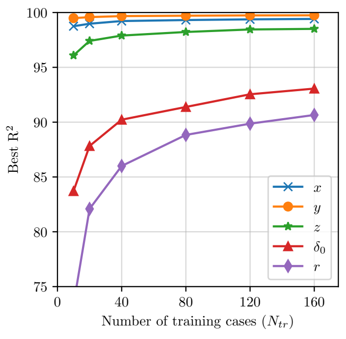
Figure 5.8: R^2 histogram for all test cases for the example with varying flow conditions.



(a) Model 1



(b) Model 2



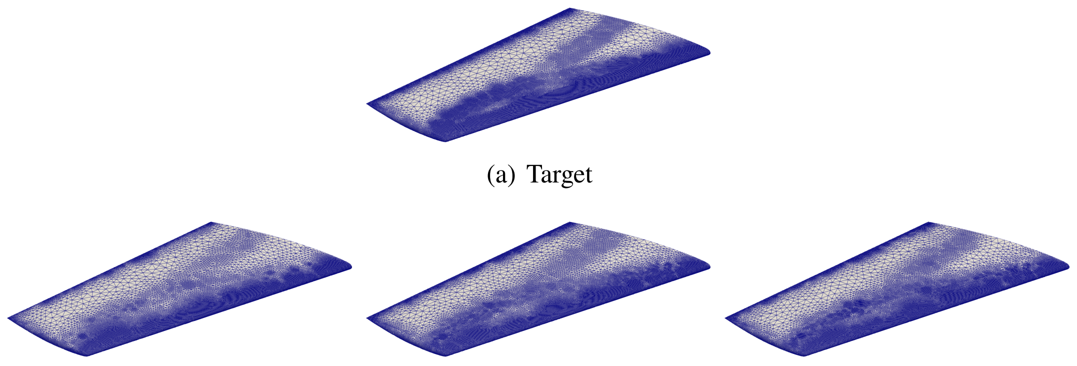
(c) Model 3

Figure 5.9: Minimum R^2 for the five mesh characteristics as a function of the number of training cases for the three models of Table 5.1.

demonstrates the accuracy of the predictions for unseen cases encompassing both subsonic and transonic flow features.

The next experiment aims to investigate the influence of the number of training cases, N_{tr}, on the accuracy of the predictions and the performance of the three models described in Table 5.1. Figure 5.9 shows the minimum R^2 for the five mesh characteristics as a function of the number of training cases, for the three models of Table 5.1. The minimum number of training cases used was 10, and it was increased until the total number of available cases was selected. All test cases were used to assess the accuracy of the selected NN regardless of the number of training cases used. For each number of training cases, the hyperparameters were tuned, as presented earlier.

The results show that very few training cases are required to perform accurate pre-


(a) Target

(b) Model 1 (c) Model 2 (d) Model 3

Figure 5.10: Target mesh and predicted meshes using the three models of Table 5.1 for $M_\infty = 0.79$ and $\alpha = 5.39°$ – the most difficult case.

dictions of the coordinates of the sources. This is expected because, as explained in Section 5.1.2, sources are grouped based on proximity. The prediction of the spacing and radius is much more challenging, and the results show that models 2 and 3 are able to outperform model 1. This is because both models 2 and 3 train an independent NN to predict the spacing and radius, whereas in the first model, a single NN is used to predict all characteristics. The performance of models 2 and 3 is almost identical.

To illustrate the potential of the proposed approach, the trained NNs are next used to predict the characteristics of the sources and near-optimal meshes are generated and compared with target meshes. All the meshes are generated following the process to eliminate redundant point sources described in Algorithm 3. In addition, point sources are merged into line sources using Algorithm 4.

Figure 5.10 shows the target mesh alongside the predictions obtained using the three models described in Table 5.1. The flow conditions correspond to $M_\infty = 0.79$ and $\alpha = 5.39°$, representing the most challenging test case. The meshes produced with the three models exhibit the refinement required to capture the most important features of this transonic flow. However, model 1 predicts a larger spacing near the location of the shock, whereas the characteristics of the sources produced by models 2 and 3 produce a spacing much more similar to the target. It is worth noting that the only distinction between Models 2 and 3 lies in the method used to predict source point locations. Model 3 employs independent neural networks for each of the three spatial coordinates,

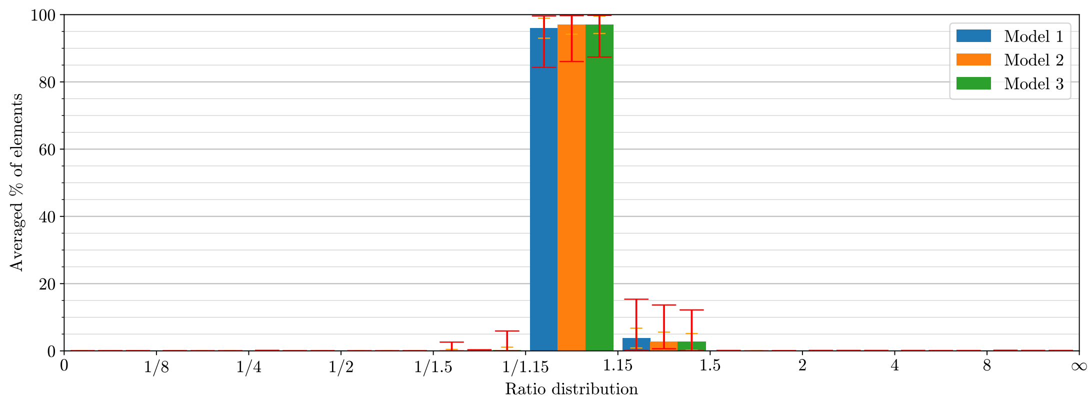
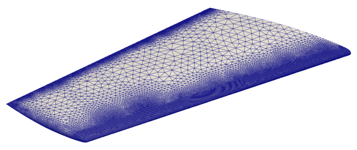


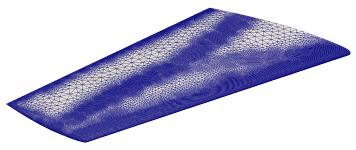
Figure 5.11: Histogram of the ratio between the predicted and target spacing for the three models of Table 5.1.

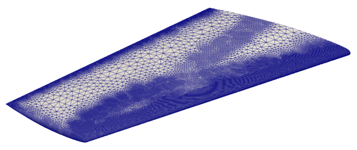
which is expected to improve the accuracy of source placement. Nevertheless, the improvement is likely to be marginal, as previous experiments have shown that source location is among the easiest quantities to predict, even with a significantly reduced training set.

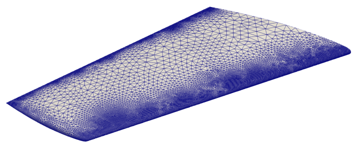
To further illustrate the performance of each model, the spacing function induced by the predicted sources is compared with the spacing function induced by the target sources. To this end, the spacing induced by the predicted sources is compared to the target spacing at the centroid of each element of the target mesh, for all test cases. Figure 5.11 shows the histogram of the ratio between predicted and target spacing for the three models. The minimum and maximum values for each bin in the histogram are depicted with red error bars, whereas the orange bar represents the standard deviation from the mean. A value of the ratio between 1/1.15 and 1.15 is considered extremely accurate and it should produce a mesh able to capture all the targeted flow features. Values of the ratio higher than 1.15 indicate that there are regions where the NN predicts a larger spacing, resulting in an under-refined mesh that can lead to missing some flow features. Similarly, a value of the ratio below 1/1.15 indicates that there are regions where the NN predicts a smaller spacing, resulting in regions unnecessarily refined.

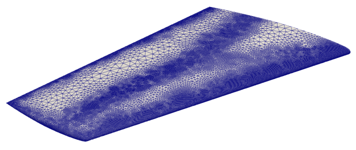
The results in Figure 5.11 show a very similar performance of the three models. Model 1 exhibits a slightly lower value in the middle bin, whereas the bin between 1.15 and 1.15 contains a larger percentage of elements when compared to models 2 and 3. It can


(a) $M_\infty = 0.41, \alpha = 8.90°$


(b) $M_\infty = 0.80, \alpha = 8.19°$


(c) $M_\infty = 0.79, \alpha = 5.39°$


(d) $M_\infty = 0.41, \alpha = 8.90°$


(e) $M_\infty = 0.80, \alpha = 8.19°$

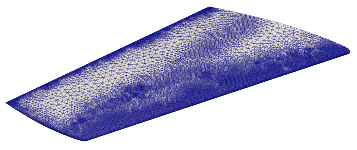

(f) $M_\infty = 0.79, \alpha = 5.39°$

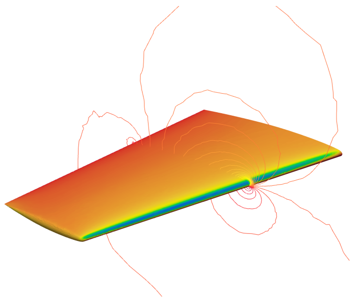
Figure 5.12: Target (top row) and predicted (bottom row) meshes using model 2 for three flow conditions.

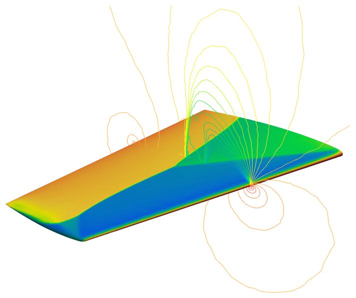
also be observed that the worst performing case for model 1 is less accurate than the worst case produced from models 2 and 3. The performance of models 2 and 3 is very similar, with a marginal better performance provided by model 3.

Using the best available NN in model 3, the predicted sources for the three test cases are used to produce the near-optimal meshes for the cases shown in Figure 5.4. Figure 5.12 shows the target and predicted meshes for three different flow conditions. These results illustrate the ability of the proposed framework to generate meshes that reflect key flow-dependent features, such as the leading edge and the presence and location of shocks. The meshes produced are qualitatively consistent with the target, showing appropriate local refinement in aerodynamically significant regions.

Some discrepancies are evident, particularly in the form of patchier meshes within the shock-dominated areas. These regions tend to be slightly under-refined compared to the target mesh, resulting in clusters of coarser element distributions. This observation is consistent with the histogram in Figure 5.24, which shows that when prediction errors occur, the model more frequently overestimates the element sizing, leading to an under-resolved mesh in those areas.

The solutions obtained by using the predicted near-optimal meshes are shown in Figure 5.13. These plots display the surface pressure coefficient, C_p, across the wing for three distinct flow conditions.


(a) $M_\infty = 0.41, \alpha = 8.90°$


(b) $M_\infty = 0.80, \alpha = 8.19°$

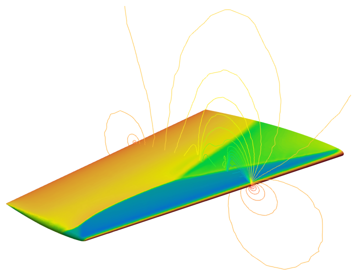
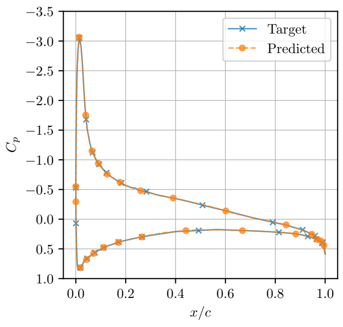
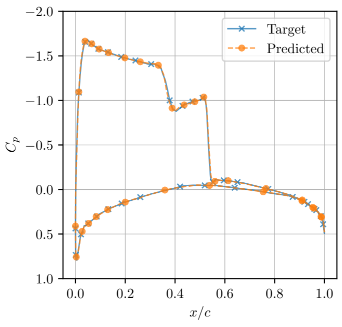

(c) $M_\infty = 0.79, \alpha = 5.39°$

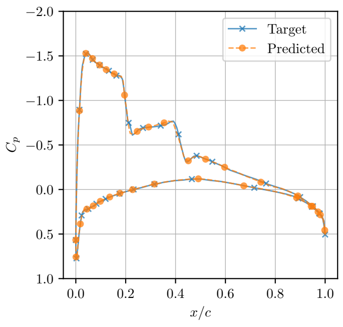
Figure 5.13: Pressure coefficient, C_p, for three different flow conditions, computed using the predicted near-optimal meshes of Figure 5.12.



(a) $M_\infty = 0.41, \alpha = 8.90°$



(b) $M_\infty = 0.80, \alpha = 8.19°$



(c) $M_\infty = 0.79, \alpha = 5.39°$

Figure 5.14: Comparison of the pressure coefficient, C_p, for three different flow conditions, at one section.

To further compare the accuracy of the solutions obtained on the predicted near-optimal meshes, Figure 5.14 compares the pressure coefficient at one section of the wing with the pressure coefficient computed with the target mesh. The results demonstrate excellent agreement between the predicted and reference solutions across all three flow regimes. The pressure distributions align closely in both magnitude and shape, including in regions with strong gradients. Notably, discontinuities – such as those associated with transonic shocks, are resolved sharply, indicating that the predicted meshes retain sufficient local resolution to capture abrupt changes in pressure. These comparisons suggest that the near-optimal meshes inferred from the neural network-based source predictions are capable of delivering solutions that match the fidelity of those obtained with conventionally generated adaptive meshes.

Finally, the lift, C_L, and drag, C_D, coefficients obtained from the simulations using the

Table 5.2: Comparison of the aerodynamic coefficients computed with the target and predicted meshes for three flow conditions.

	$M_\infty = 0.41, \alpha = 8.90°$		$M_\infty = 0.80, \alpha = 8.19°$		$M_\infty = 0.79, \alpha = 5.39°$	
	Target	Prediction	Target	Prediction	Target	Prediction
C_L	0.605	0.603	0.722	0.723	0.469	0.468
C_D	0.0342	0.0340	0.0828	0.0828	0.0289	0.0287

meshes generated from the target sources and the meshes generated from the predicted sources are compared in Table 5.2. A maximum difference of two lift counts and two drag counts, further confirms the ability of the trained sources to construct the required meshes for unseen cases.

An alternative approach to using a NN to predict a near-optimal mesh for an unseen test case would be to use an optimal mesh from a similar flight condition from the training dataset. A case from the unseen test set is selected to compare this alternative method with the NN approach presented in this work. The case used is a transonic case, where the optimal mesh varies significantly as the parameters change, and the accuracy of the solution is very sensitive to using an appropriate mesh.

Taking an example flight condition of (M_∞, α) being $(0.79, 5.39°)$, above, the *nearest* training cases in the parametric space corresponds to (M_∞, α) equal to $(0.82, 5.34°)$. Taking the optimal mesh from the training case and using it on the unseen test flight condition, a solution using the given mesh is obtained.

Comparing the lift, C_L, and drag, C_D, coefficients obtained using the two methods, a very large difference is found between the expected target results and the obtained results using a mesh from a similar case. The difference being 8 lift counts and 12 drag counts, which is beyond what is to be tolerated. Whereas the NN could accurately predict a mesh able to capture the solution within one lift and drag count of the target solution.

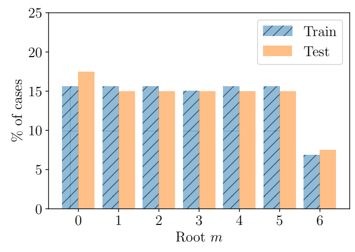
5.2.2 Near-optimal mesh predictions on a variable wing geometry at fixed inflow condition

The second example involves the prediction of near-optimal meshes for a geometrically parametrised wing at a fixed transonic condition of $M_\infty = 0.85$ and $\alpha = 3°$. The geometry is constructed from two different four-digit NACA aerofoils placed at the root and the tip of the wing. A linear variation of the geometry is considered in the span direction of the wing. The three parameters of each four-digit NACA aerofoil form the input of the NN. Given the better performance of the model 3 architecture demonstrated in the previous example, this and the following example in Section 5.2.3, only use this model type for the NNs.

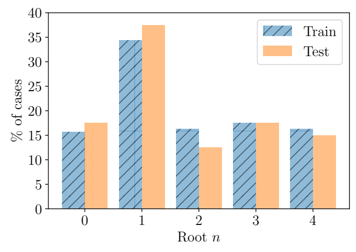
Halton sequencing of the six input parameters is used to generate a training dataset consisting of $N_{tr} = 160$ training cases. For both aerofoils, the range of the maximum camber, m, is taken between 0 and 6, the range of location of the maximum camber, n, is set between 0 and 4 (corresponding to 0% and 40% of the chord) and the range of the thickness, p, is set between 6 and 24. A further dataset of $N_{tst} = 40$ test cases is also generated using Halton sequencing. As in the previous example, the range of inputs used to generate the test cases is slightly modified to minimise the undesired use of the trained NNs for extrapolation. To illustrate the training and test datasets employed, Figure 5.15 displays a histogram of the two datasets.

For each training and test case, the CFD solution is obtained using FLITE (Sørensen et al., 2003a) on an unstructured tetrahedral mesh consisting of approximately 1.3M elements and 250K nodes. Figure 5.16 shows the pressure coefficient, C_p, on the surface and a cut across the wing for three geometries from the test set. The number of sources generated from the solution of each case in the datasets, based on Algorithm 1, varied between 3,949 and 7,571. The global set of sources that resulted from the combination of sources described in Algorithm 2, and used for training, consisted of 48,503 sources.

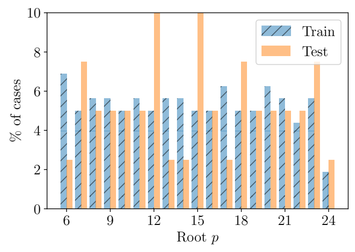
Following the rationale of the previous example, the influence of the number of training



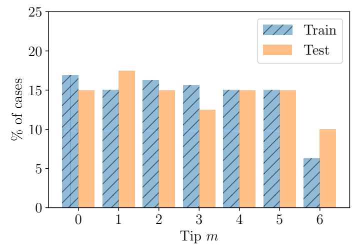
(a)



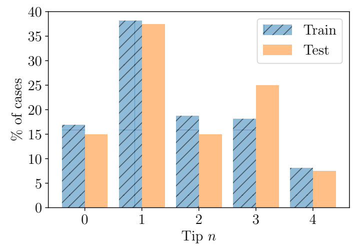
(b)



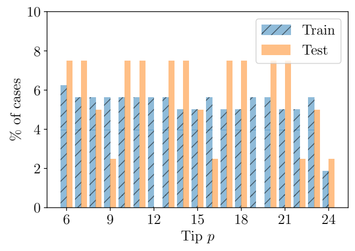
(c)



(d)

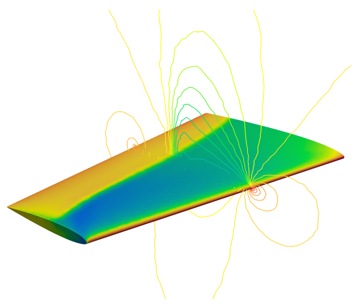


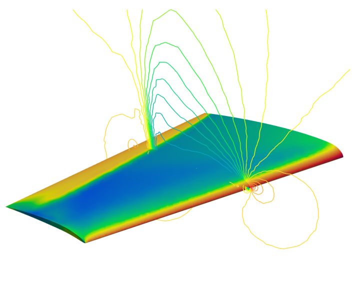
(e)



(f)

Figure 5.15: Histograms of the sampling data used for the training and test sets of the variable wing geometry.


(a) NACA0010–NACA1115


(b) NACA4109–NACA6306

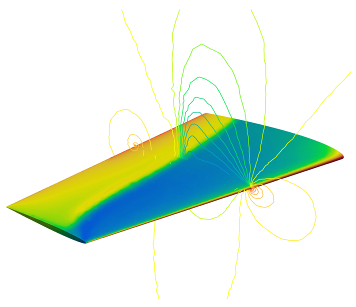

(c) NACA0012–NACA0007

Figure 5.16: Pressure coefficient, C_p, for three different geometric configurations. For each case the first NACA corresponds to the root of the wing and the second NACA corresponds to the tip.

cases in the accuracy of the predictions is studied. For each number of training cases, the NN architecture is tuned by varying the number of hidden layers and neurons in each layer. Figure 5.17 shows the minimum R^2 for the five mesh characteristics as a function of the number of training cases. Similar to the previous example, predicting the coordinates of the sources requires very few training cases to achieve an excellent predicting accuracy. Predicting the spacing and radius of the sources is more difficult, but with 40 training cases the lowest R^2 is already above 80. It is worth noting that the same level of R^2 achieved in the first example is not reached. This is mainly attributed

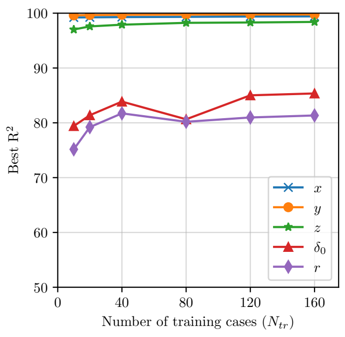


Figure 5.17: Minimum R^2 for the five mesh characteristics as a function of the number of training cases for the example with varying geometry.

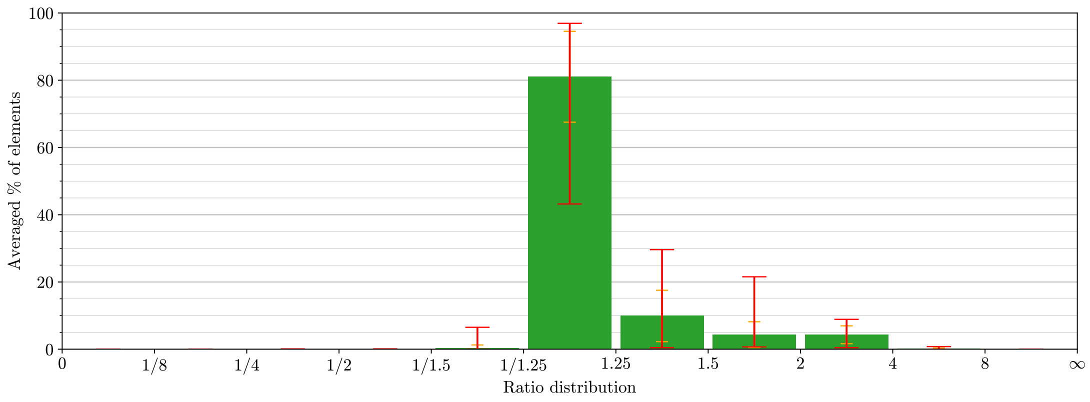
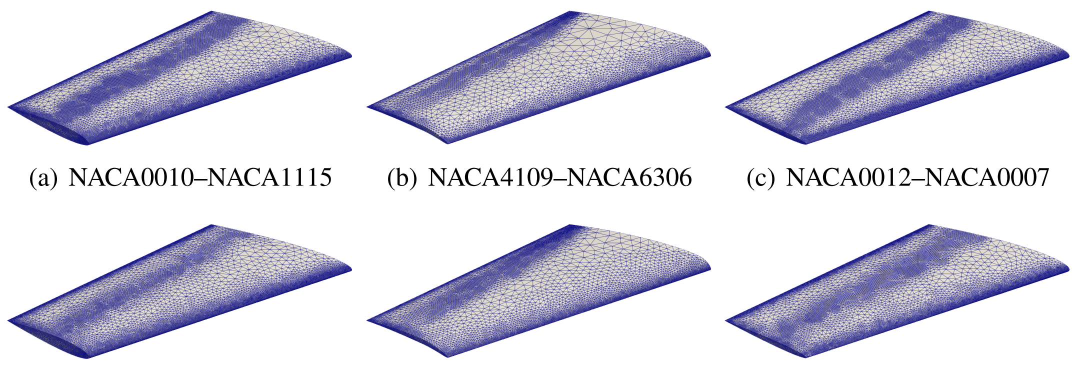


Figure 5.18: Histogram of the ratio between the predicted and target spacing for the example with varying geometry.

to the fact that in both cases the maximum number of training cases is 160 but this example contains three times more parameters. In addition, it is worth noting that the parameters considered in this example are geometric parameters and it is usually more difficult to generate reduced order models, when compared to flow conditions (Sevilla et al., 2020; Balla et al., 2021).

To quantify the ability of the model to predict the correct characteristics of the sources that produces a mesh comparable to the target mesh, the ratio between the spacing computed using the predicted sources and the target sources at the centroid of the elements was evaluated. Figure 5.18 shows the histogram of ratio between predicted and target spacing. The results show that, despite the reduced size of the training dataset, less than 5% of the elements generated for all test cases have a spacing more than double the target spacing. The results suggest that more training cases that better sample the parametric space are needed. This can be achieved by using a more generic



(a) NACA0010–NACA1115 (b) NACA4109–NACA6306 (c) NACA0012–NACA0007

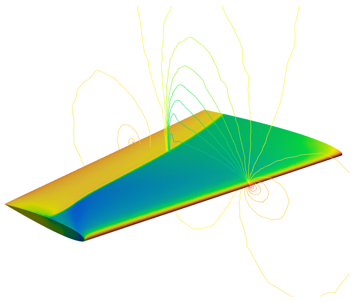
(d) NACA0010–NACA1115 (e) NACA4109–NACA6306 (f) NACA0012–NACA0007

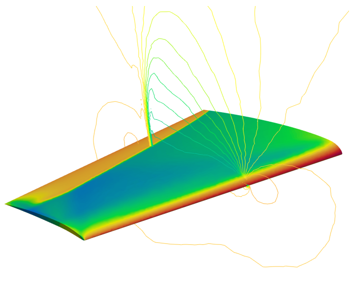
Figure 5.19: Target (top row) and predicted (bottom row) meshes for three geometric configurations.

definition of the geometry and eliminating the need to have the NACA digits as integer values.

Using the best available NN, the predicted sources are used to produce the near-optimal meshes for the three test cases outlined in Figure 5.16. Figure 5.19 shows the target and predicted meshes for the three different geometric configurations. Despite the low number of training cases used for this problem involving six geometric parameters, the proposed approach is able to predict near-optimal meshes, capturing the local refinement required for different geometric configurations. It is worth noting that the last geometric case considered in Figure 5.19 corresponds to a point near the boundary of the six-dimensional parametric space and therefore the accuracy of the prediction is expected to be lower, when compared to other cases.

The solutions obtained by using the predicted near-optimal meshes are shown in Figure 5.20 . To better compare the accuracy of the solutions obtained on predicted near-optimal meshes, Figure 5.21 compares the pressure coefficient, at one section of the wing, with the pressure coefficient computed with the target mesh. The comparison of pressure coefficients shows that, despite some differences can be observed on the predicted meshes of Figure 5.19, the predicted near-optimal meshes are capable of capturing all the flow features. To confirm this finding, the lift, C_L, and drag, C_D, coefficients obtained from the simulations using the meshes generated from the target


(a) NACA0010–NACA1115


(b) NACA4109–NACA6306

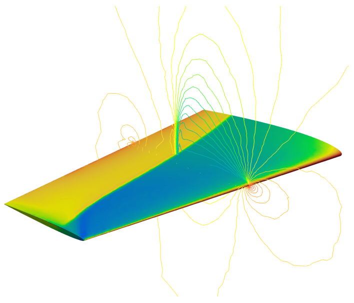
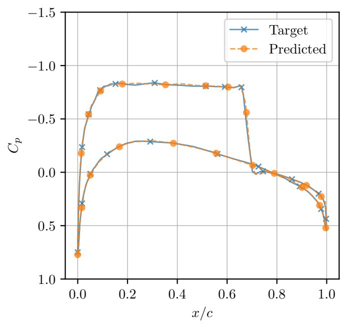
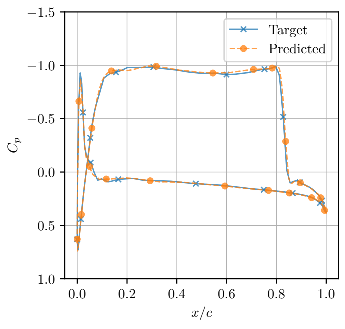

(c) NACA0012–NACA0007

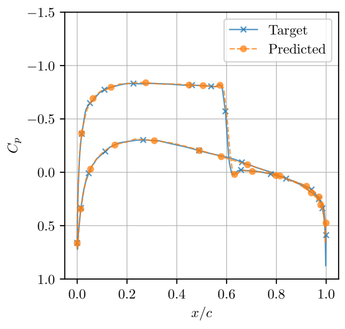
Figure 5.20: Pressure coefficient, C_p, for three different geometric configurations, computed using the predicted near-optimal meshes of Figure 5.19.



(a) NACA0010–NACA1115



(b) NACA4109–NACA6306



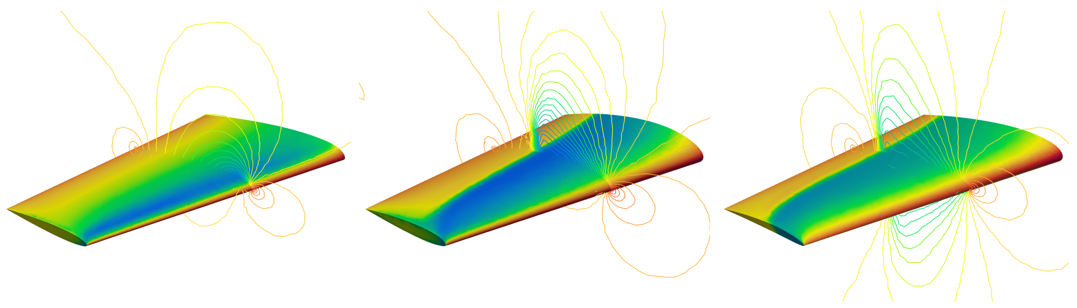
(c) NACA0012–NACA0007

Figure 5.21: Comparison of the pressure coefficient, C_p, for three different geometric configurations, at one section.

Table 5.3: Comparison of the aerodynamic coefficients computed with the target and predicted meshes for three geometric configurations.

	NACA0010–NACA1115		NACA4109–NACA6306		NACA0012–NACA00072	
	Target	Prediction	Target	Prediction	Target	Prediction
C_L	0.377	0.377	0.751	0.754	0.311	0.312
C_D	0.0275	0.0273	0.0851	0.0853	0.0154	0.0154

sources and the meshes generated from the predicted sources are compared in Table 5.3. A maximum difference of only three lift counts and two drag counts, confirms the ability of the trained sources to construct the required meshes for unseen cases.



(a) NACA3114–NACA1113, $M_\infty = 0.69, \alpha = 3.00°$ (b) NACA5421–NACA2116, $M_\infty = 0.74, \alpha = 5.60°$ (c) NACA2121–NACA4409, $M_\infty = 0.86, \alpha = -3.10°$

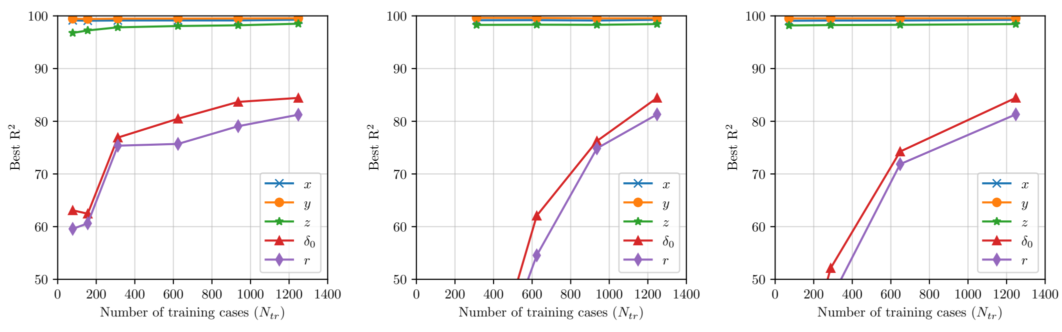
Figure 5.22: Pressure coefficient, C_p, for three different geometric configurations and flow conditions. For each case the first NACA corresponds to the root of the wing and the second NACA corresponds to the tip.

5.2.3 Near-optimal mesh predictions at various inflow conditions on a variable wing geometry

The last example involves the prediction of near-optimal meshes for a geometrically parameterised wing at variable flow conditions. The geometry is the same used in the example of Section 5.2.2, whereas the flow conditions consider a Mach number and angle of attack between $[0.6, 0.9]$ and $[-4°, 10°]$, respectively.

A Halton sequencing of the six geometric parameters is used to generate a dataset consisting of 52 cases. For each value of the geometric parameters, a Halton sequencing of the two flow parameters was used to generate a dataset that consist of 24 cases. The combination of the two datasets provides the final dataset that consisted of $N_{tr} = 52 \times 24 = 1,248$ training cases. The same procedure was used to generate a test dataset that contains $N_{tst} = 8 \times 24 = 192$ cases.

For each case in the training and test datasets, the solution is obtained by using the FLITE (Sørensen et al., 2003a) solver and an unstructured tetrahedral mesh consisting of approximately 1.3M elements and 250K nodes. The distribution of the pressure coefficient for three test cases is shown on Figure 5.22. The number of sources generated from the solution of each case in the datasets, based on Algorithm 1, varied between 1,533 and 7,582. The global set of sources that resulted from the combination



(a) Increasing geometry (b) Increasing flow conditions (c) Increasing both

Figure 5.23: Minimum R^2 for the five mesh characteristics as a function of the number of training cases for the example with varying flow conditions and geometry. Three different strategies of increasing the number of training cases are considered.

of sources described in Algorithm 2, and used for training, consisted of 54,713 sources.

The influence of the number of training cases in the accuracy of the predictions is studied next. To better understand the importance of the flow and geometric parameters separately, the increase of the number of training cases is performed in three different ways. First, the number of training cases are increased by increasing only the number of geometric configurations, starting with 3 cases and doubling the number of geometric cases until all geometric configurations available are considered. For each geometric configuration, the 24 available flow conditions were used. Second, the number of training cases are increased by increasing only the number of flow conditions considered. Finally, the increase in number of training cases is performed by not distinguishing the type of parameters, so increasing both the number of flow conditions and geometric configurations considered.

Figure 5.23 shows the minimum R^2, for the five mesh characteristics, as a function of the number of training cases. The qualitative behaviour is similar to the previous examples. With a very reduced dataset for eight parameters, it is possible to produce very accurate predictions of the coordinates of the sources, whereas accurate predictions of the spacing and radius require a significant increase in the number of training cases. For a given number of training cases, the best NN architecture was found by varying the number of hidden layers and neurons in each layer, following the same process

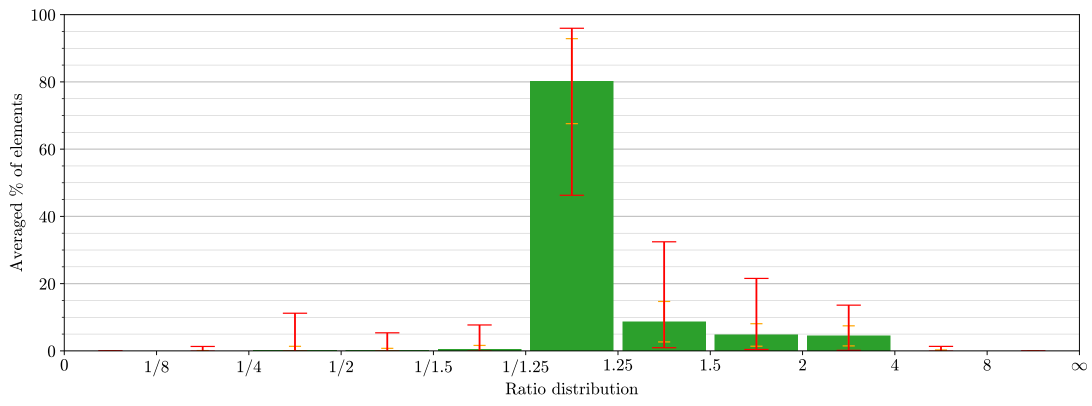
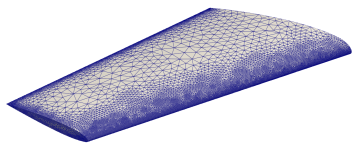


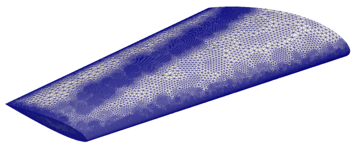
Figure 5.24: Histogram of the ratio between the predicted and target spacing for the example with varying flow conditions and geometry.

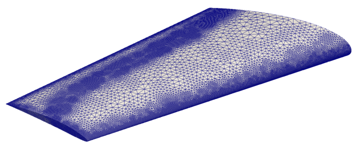
described in previous examples.

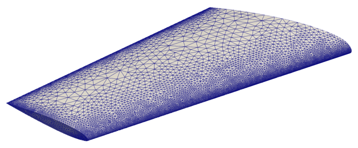
To quantify the ability of the model to predict the correct characteristics of the sources that produces a mesh comparable to the target mesh, the ratio between the spacing computed using the predicted sources and the target sources at the centroid of the elements of the target mesh is evaluated. Figure 5.24 shows the histogram of the ratio between predicted and target spacing. Despite the larger number of parameters and the different nature of the parameters involved, the results show that less than 5% of the elements generated for all test cases have a spacing more than double the target spacing.

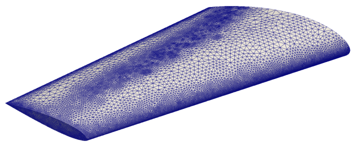
By using the predicted characteristics of the sources, meshes for the three test cases shown in Figure 5.22 are produced, and compared to the meshes obtained with the target characteristics. As in the previous examples, all the meshes are generated following the process to eliminate redundant point sources described in Algorithm 3 and merging point sources into line sources by using Algorithm 4. Figure 5.25 shows the target and predicted meshes for the three different geometric configurations. The results show that the proposed approach is able to predict the characteristics of the sources in such a way that the generated meshes provide appropriate mesh resolution near the regions where it is needed. In the first example (left), the predicted mesh under-resolves the leading edge on the upper surface, resulting in a slightly coarser resolution than the target in this region. In the second case (centre), the predicted mesh captures the shock location with good accuracy but exhibits a more blotchy pattern in the refined


(a) NACA3114–NACA1113, $M_\infty = 0.69, \alpha = 3.00°$


(b) NACA5421–NACA2116, $M_\infty = 0.74, \alpha = 5.60°$


(c) NACA2121–NACA4409, $M_\infty = 0.86, \alpha = -3.10°$


(d) NACA3114–NACA1113, $M_\infty = 0.69, \alpha = 3.00°$


(e) NACA5421–NACA2116, $M_\infty = 0.74, \alpha = 5.60°$

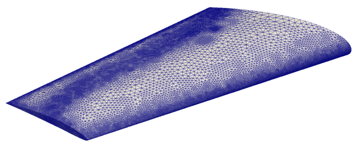
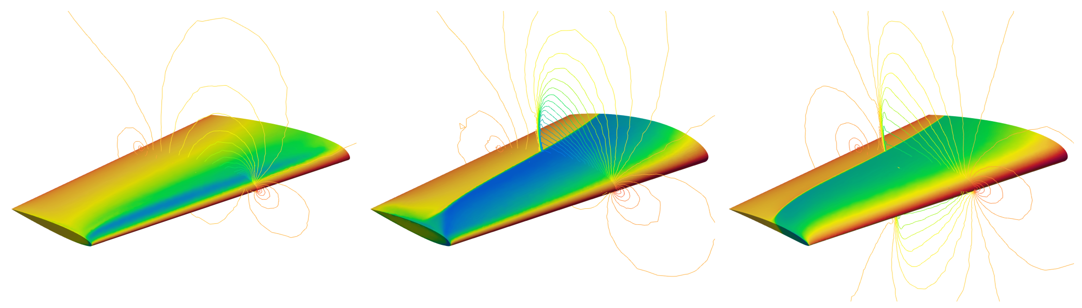

(f) NACA2121–NACA4409, $M_\infty = 0.86, \alpha = -3.10°$

Figure 5.25: Target (top row) and predicted (bottom row) meshes for three flow conditions and geometric configurations.

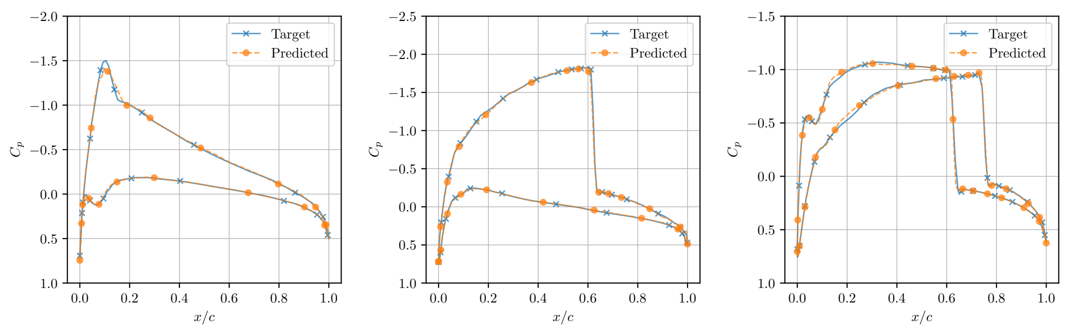
region, possibly due to small inaccuracies in the predicted source strengths or overlap. The third case (right) demonstrates close agreement between the predicted and target mesh structures; however, a localised over-refinement is visible ahead of the shock on the upper surface, likely caused by a mispredicted source introducing unnecessary small spacing in that region. Despite these local differences, the overall structure and flow-aware characteristics are preserved, confirming that the model captures the essential spacing features across varied geometric configurations. The smoothness of the spacing function in the second and third test cases of Figure 5.25 could be improved by increasing the number of training cases or by modifying the similarity tolerance used for grouping sources.

To analyse the ability of the near-optimal meshes generated to capture the required flow features, Figure 5.26 shows the pressure coefficient distribution obtained with the predicted meshes of Figure 5.25. The first case (left), which corresponds to the subsonic flow regime, exhibits smooth pressure variation with no shocks present. The pressure field is well-resolved, with accurate capture of the suction peak near the leading edge, demonstrating that the mesh density is sufficient to resolve key aerodynamic gradients. In the second case (centre), the solution features a strong shock on the upper surface, which is sharply resolved by the predicted mesh, indicating that the spacing has been well targeted. The shock discontinuity is narrow, however, the flow near the wing tip



(a) NACA3114–NACA1113, $M_\infty = 0.69, \alpha = 3.00°$ (b) NACA5421–NACA2116, $M_\infty = 0.74, \alpha = 5.60°$ (c) NACA2121–NACA4409, $M_\infty = 0.86, \alpha = -3.10°$

Figure 5.26: Pressure coefficient, C_p, for three different flow conditions and geometric configurations, computed using the predicted near-optimal meshes of Figure 5.26.



(a) NACA3114–NACA1113, $M_\infty = 0.69, \alpha = 3.00°$ (b) NACA5421–NACA2116, $M_\infty = 0.74, \alpha = 5.60°$ (c) NACA2121–NACA4409, $M_\infty = 0.86, \alpha = -3.10°$

Figure 5.27: Comparison of the pressure coefficient, C_p, for three different flow conditions and geometric configurations, at one section.

appears slightly more diffused, likely due to localised under-refinement in that region. The third case (right) also features a strong shock on the upper surface and demonstrates excellent agreement with expected flow behaviour. The shock is cleanly resolved, and the flow features are preserved across the domain, confirming the suitability of the predicted mesh for high-speed conditions involving complex geometry and strong gradients.

To better quantify the accuracy of the solutions obtained on predicted near-optimal meshes, Figure 5.27 compares the pressure coefficient, at one section of the wing, with the pressure coefficient computed with the target mesh. It can be clearly observed that despite the lack of smoothness in the spacing functions obtained with the predicted characteristics of the sources, the near-optimal meshes are able to correctly capture all

Table 5.4: Comparison of the aerodynamic coefficients computed with the target and predicted meshes for three geometric configurations.

	NACA3314–NACA1113 $M_\infty = 0.69$, $\alpha = 3.00°$		NACA5421–NACA2116 $M_\infty = 0.74$, $\alpha = 5.60°$		NACA2121–NACA4409 $M_\infty = 0.86$, $\alpha = -3.10°$	
	Target	Prediction	Target	Prediction	Target	Prediction
C_L	0.435	0.436	0.805	0.804	-0.021	-0.020
C_D	0.0158	0.0162	0.1009	0.1013	0.0509	0.0508

the required flow features, offering an excellent agreement with the solutions computed on the target meshes.

Finally, the lift and drag coefficients resulted from the simulation using the mesh generated from target sources and the mesh generated from the predicted sources are compared in Table 5.4. A maximum difference of only one lift count and four drag counts confirms the ability of the trained NNs to predict the characteristics of the mesh sources for unseen cases.

5.3 Efficiency and environmental implications

The strategy proposed in this work relies on training a relatively large number of NNs using deep learning. The accuracy of the predictions has been evaluated and the implications in terms of reducing the number of hours required to produce near-optimal meshes for simulation are clear. However, in the last few years there have been growing concerns about the environmental impact of training large models using NNs (Schwartz et al., 2020; Lannelongue et al., 2021). This is because the majority of articles found in the literature tend to focus on the accuracy of the predictions, but ignore the resources required to train and tune the NNs.

This Section aims at analysing the efficiency of the proposed approach to demonstrate that the strategy presented can be labelled as *Green AI research* (Schwartz et al., 2020), in the sense that it reduces the computational cost that is currently required in industry

Table 5.5: Details of the algorithm and HPC facilities used.

Type of core	CPU
Number of cores (Mesh generation)	1
Number of cores (CFD)	24-64
Number of cores (NN)	1
CPU Model	Intel® Xeon® Gold 6252
Memory available	96GB-192GB
Platform	Local server
Geographical location	United Kingdom
Real CPU usage factor	1.0
Power usage efficiency (PUE)	1.4

to achieve similar results.

When considering the computational and environmental impact of a methodology based on deep learning, several factors must be taken into consideration. These factors include not only the training time for one model, but also the time required to perform the hyper-parameter tuning. In addition, other factors such as the computing infrastructure used, the hardware architecture or even the geographical location of the high-performance computing (HPC) facilities employed (Lacoste et al., 2019; Lannelongue et al., 2021).

In this work, the model developed in (Lannelongue et al., 2021) to estimate the carbon footprint of a computation is considered. The characteristics of the HPC facilities used in all the computations presented in this work are summarised in Table 5.5. The algorithms involved, when measuring the efficiency of the proposed approach, are the mesh generation algorithm, the CFD solver and the NN training. The mesh generator runs on a single processor. The CFD solver runs on 24 processors for meshes between 5M to 6.5M elements and on 64 processors for a mesh of 40M elements. Finally, the training of the NN is performed using a single processor.

To analyse the efficiency and environmental impact of the methodology proposed in this work, the common task of performing CFD computations for a varying free-stream Mach number and angle of attack is considered. This corresponds to the first example described in Section 5.2.1. The free-stream Mach number varies between 0.3 and 0.9,

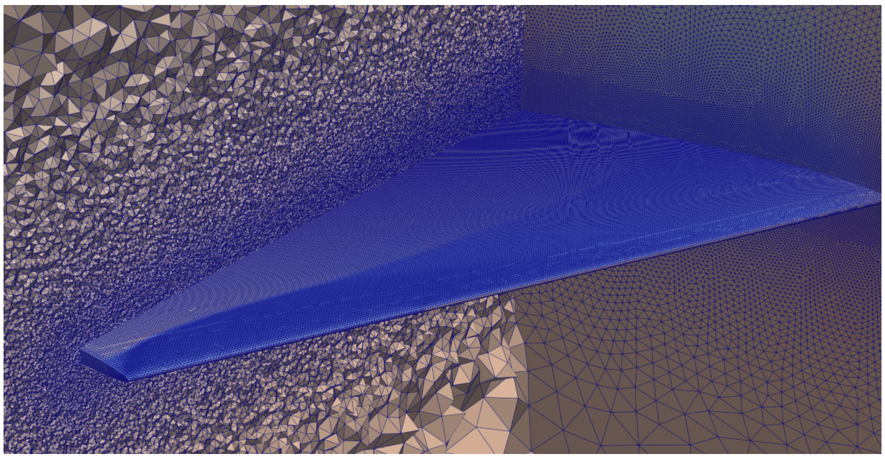


Figure 5.28: Fine mesh around the M6 wing.

and computations are performed in steps of 0.05. The angle of attack varies between -4° and 12°, and computations are performed in steps of 1.6°. This means that a total of 143 CFD computations are required.

The most common approach considered in industry when performing this parametric study consists of generating a fixed, very fine, mesh that is capable of capturing the solution for all the cases of interest (Michal, 2019). This is done in practice to minimise the time-consuming task of generating a mesh tailored for every single case, which is not only difficult, but actually requires a significant amount of human intervention. For the example considered here, an unstructured mesh of 40M tetrahedral elements is generated. A detailed view of the fine mesh generated is shown in Figure 5.28. The generation of this mesh takes only one hour on a single processor. However, the implications of generating a single mesh are substantial when running the CFD solver. In this example, each CFD simulation takes 24 hours using 64 processors, which implies a server memory capacity of 192GB. This amounts for a total of 3,432 hours to compute the 143 solutions required.

Table 5.6 summarises the carbon footprint and energy consumption induced by the process of computing the 143 solutions with a fixed mesh. The footprint is obviously dominated by the CFD calculations, and the total amount is equivalent to more than 3,000km in a passenger car, or almost a flight from New York City to San Francisco (Lannelongue et al., 2021).

Table 5.6: Carbon footprint and energy consumption for the parametric study using a fixed mesh capable of accurately capturing all the solutions.

Task	Wall clock (H)	Carbon (Kg CO_2e)	Energy (MWh)
Mesh generation	1.0	3.61×10^{-3}	5.89×10^{-5}
CFD solution	3,432.10	527.17	2.28
Total	3,433.0	527.17	2.28

Table 5.7: Carbon footprint and energy consumption for the parametric study using the proposed approach to train a NN, including tuning the hyperparameters, predict the near-optimal meshes and run the CFD simulations.

Task	Wall clock (H)	Carbon (Kg CO_2e)	Energy (MWh)
NN tuning and training	156.6	2.13	9.22×10^{-3}
Mesh generation	23.8	0.32	1.40×10^{-3}
CFD solution	143.0	12.36	5.35×10^{-2}
Total	323.4	14.81	0.064

The alternative, proposed in this work, is to train a NN with all the data that is currently available in industry and use the predictions to generate tailored near-optimal meshes for each case. With this strategy the mesh generation stage is no longer time consuming and does not require the input of an expert engineer. In the current example, each near-optimal mesh is generated in 10 minutes using a single processor. The near-optimal meshes have between 5M and 6.5M elements for all the cases considered, which is a fraction of the number of elements required for the fixed-mesh approach currently employed in practice. For the near-optimal meshes, the CFD simulations run on 24 processors, with 96GB available memory, and the solution requires between 40 minutes and one hour and 20 minutes. It is worth noting that the variation of the time required for the solution is induced by the variation in the mesh that is required for each case.

Table 5.7 summarises the carbon footprint and energy consumption induced by the proposed approach. This includes the resources required to tune the hyperparameters of the network, the generation of 143 near-optimal meshes and the associated CFD simulations. It is worth noting that the time required for tuning accounts for the fact that the experiments to tune the NNs were repeated five times, to minimise the effect of the random initialisation of the NN weights.

Whilst the carbon footprint associated with the CFD simulations constitutes the majority of the total emissions, the contribution from neural network tuning and training, though significantly smaller, is still notable. This aspect is particularly noteworthy given that the tuning stage alone accounts for over 14% of the total footprint, yet is often overlooked in similar studies (Schwartz et al., 2020). Despite employing a grid search strategy for hyperparameter selection and repeating each training run five times for statistical robustness, the total carbon footprint and energy usage of the proposed method remain more than 35 times lower than those associated with the conventional approach of running all cases on a fixed, over-refined mesh.

It is worth noting that it is possible to reduce the computational cost of performing the simulations in a very fine mesh by utilising the solution with a certain Mach number and/or angle of attack for another case. However, it is very important to note that the proposed approach also permits this type of solution restarting after interpolating one computed solution to a new predicted mesh. Therefore, the gain is expected to be very similar to the one reported here. In addition, it is worth mentioning that on some occasions, this restarting approach is not preferred because it requires running all the cases serially.

5.4 Concluding Remarks

This chapter has successfully demonstrated the capability of using neural networks to predict mesh spacing fields directly from flow and geometry parameters. By learning the underlying relationship between configuration inputs and mesh control parameters, the method enables the generation of high-quality meshes without requiring access to the corresponding flow solution. The implementation explored in this chapter employs a source-based formulation, wherein the spacing field is encoded using a set of isotropic point sources. Applied to the ONERA M6 wing, this approach allowed for the accurate prediction of source characteristics from free-stream and geometric inputs, facilitating the automated construction of unstructured meshes suitable for steady-state inviscid simulations. Inference is computationally inexpensive and requires minimal architectural tuning, making the method well-suited for use in parametric studies or design optimisation pipelines.

Three neural network models were evaluated, each differing in how the source characteristics were predicted. The results confirmed that separating the prediction of spacing and radius from that of the spatial coordinates improves accuracy – particularly for the more demanding transonic cases. The predicted meshes led to flow solutions closely aligned with those obtained using reference meshes, with differences in lift and drag remaining within two counts – well within accepted engineering tolerances.

A key strength of the proposed source-based approach is its flexibility. The framework developed in this chapter, based on generating sources independently for each configuration and subsequently combining them into a unified global set naturally accommodates geometric variation. This decoupled structure allows for straightforward extension to scenarios involving complex geometric changes, without imposing restrictions on the nature or scale of the variation. By treating each configuration in isolation prior to merging, the method preserves local mesh accuracy while supporting a consistent and scalable training strategy.

However, a limitation of the source-based method lies in its restricted capacity to represent anisotropic refinement. While point sources offer an efficient means of defining isotropic spacing fields, they afford limited directional control, which can be essential in accurately resolving flow-aligned features such as shocks. Additionally, although each source describes a region of the domain – thereby reducing the total number of points requiring prediction, the need for five separate characteristics per source can result in a more complex representation of the spacing field than may be necessary. In some cases, a simpler formulation based on discrete spacing values may offer a more efficient and interpretable alternative. The next chapter builds upon this work by introducing a background mesh formulation, which offers a simplification by requiring only a single spacing value per point, offering a more compact and scalable formulation.

Chapter 6

Predicting the Near-Optimal Mesh Using a Background Mesh

This chapter investigates two strategies for predicting near-optimal mesh spacing fields using neural networks: one based on mesh sources, and another based on a background mesh. The work presented here is based on the paper ”*Predicting the Near-Optimal Mesh Spacing for a Simulation Using Machine Learning*,” published in the proceedings of the *SIAM International Meshing Roundtable 2023* (Lock et al., 2024).

The methodology for the source-based strategy has been introduced in Chapter 5. In this chapter, the focus is on presenting a new approach in which the spacing is defined over a background mesh. A conservative interpolation method is proposed to transfer spacing from high-resolution solution meshes onto a coarser, fixed background mesh, producing data suitable for training.

Neural networks are then trained to predict either the spacing at the background mesh nodes or, for comparison, the previously defined source characteristics. The two strategies are evaluated in terms of training efficiency, prediction accuracy, and data requirements.

Two examples are considered. The first involves near-optimal mesh prediction for a wing under varying flow conditions and serves as a direct comparison between the two strategies. The second applies the background mesh method to a full aircraft configuration, demonstrating its robustness and scalability in more complex simulations.

The chapter concludes with a discussion of the results, highlighting the advantages of the background mesh approach in terms of simplicity, performance, and practical implementation.

6.1 Target Spacing

The method used in this work to construct the target spacing field is based on the approach introduced in Section 5.1.1. Rather than learning directly from the underlying meshes used to compute the solutions, the methodology derives spacing information from the solution fields themselves. This allows the spacing to reflect the actual features of the solution, independent of the numerical scheme or mesh that produced it.

As discussed previously, the approach employs an error-based analysis that relates the desired spacing to the second-order derivatives of a key solution variable. A recovery process is used to evaluate these derivatives numerically, from which directional spacing values are computed. The final scalar spacing at each point is extracted from these values, subject to user-defined bounds to ensure numerical robustness.

At this stage, a discrete representation of the spacing function is obtained. However, for each solution available, the number of mesh nodes is generally different, so two strategies are considered to homogenise the data in such a way that is suitable for training a NN. The first strategy, laid out in Section 5.1.2 consists of building a global set of sources that is capable of describing the spacing function of each case. The second approach proposed, consists of building a spacing function by using a background mesh that is also suitable to describe the spacing function of each case. The two strategies are described in the next two sections.

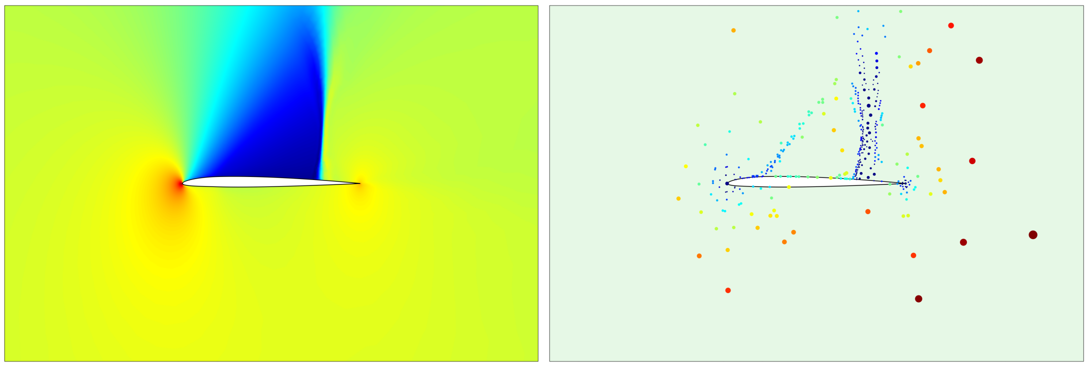

(a) Solution (b) Point sources

Figure 6.1: Transonic flow CFD solution and point sources to create a spacing function capable of capturing the solution for a NACA1206.

6.2 Spacing description using sources

In this work, the construction of a spacing function using sources follows the methodology introduced in Section 5.1.1. The same approach is adopted here, whereby local spacing requirements, obtained from the solution field, are represented using a set of point sources. These sources define spatially varying spacing functions capable of capturing key flow features while maintaining efficiency. A key aspect of the methodology is the generation of a global set of sources which provides a consistent structure across different cases, enabling their use in neural network training. The full algorithmic details and illustrative examples of this process are previously discussed in Section 5.1.2.

Figure 6.1 shows the result of creating sources to represent the spacing required to capture a given solution. The solution corresponds to an actual two-dimensional inviscid transonic flow simulation. It can be observed how the sources with smaller spacing (blue colour) are concentrated near the regions with steep gradients.

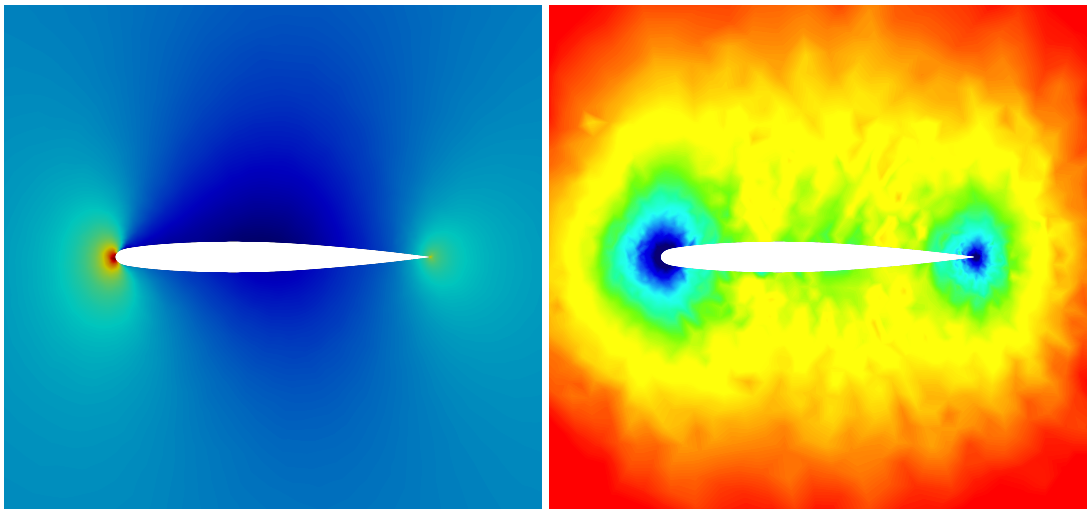

(a) Solution (b) Spacing function

Figure 6.2: A solution and its corresponding calculated spacing function that describes the optimal spacing suitable for capturing the solution.

6.3 Spacing description using a background mesh

A novel approach to building a spacing function that ensures uniformity of the data and, therefore, its possible use for training an NN is presented here. The process consists of creating a coarse background mesh and to devise a strategy to compute the spacing at each node of the background mesh that induces the required spacing to capture a given solution.

First, the spacing is computed on the mesh where the solution is provided. Figure 6.2 shows a solution and the spacing function that will provide the required mesh to capture the solution.

The implementation of this strategy introduces several advantages when compared to the existing strategy of using sources. First, the computation of the spacing at the nodes of the background mesh is simpler than the computation of the sources as it only requires interpolating the spacing from a fine mesh to a coarse mesh. This is described in detail in this section. Second, uniformity of the data is guaranteed if the topology of the background mesh is unchanged. For the examples considered in this work, with design parameters not affecting the geometry of the domain, a fixed background mesh

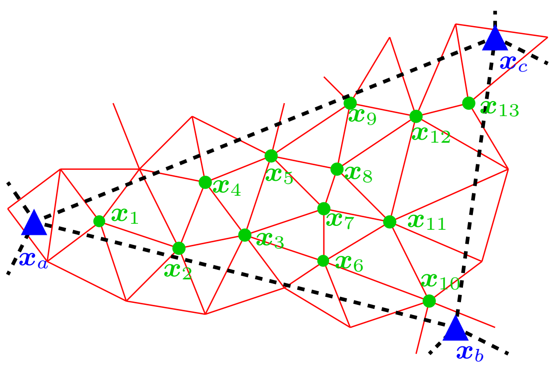


Figure 6.3: Detail of two triangular meshes, the fine mesh where the solution is computed – denoted by continuous red edges, and a coarser background mesh denoted by discontinuous black edges. The green circles denote the nodes of the fine mesh contained in one element of the background mesh. The blue triangles denote the nodes of the background mesh where the interpolated element spacing is to be computed.

can be used for all cases. For more complex scenarios, a mesh morphing algorithm would be required to ensure that the same background mesh can be used for all cases. This is out of the scope of the current work.

The fixed background mesh is produced using a combination of curvature control and minimum spacing defined at each individual surface of the geometry; an octree is then used to propagate the spacing into the domain.

6.3.1 Interpolating the spacing on a background mesh

As mentioned above, the proposed strategy to use a background mesh requires interpolating the discrete spacing from a mesh where a solution is available to a coarse background mesh. Interpolating a field from one mesh to another is a relatively easy task but special care must be taken when the quantity to be interpolated is the spacing. If a naïve interpolation is employed, many features of the solution can be unresolved by the spacing function embedded in the background mesh.

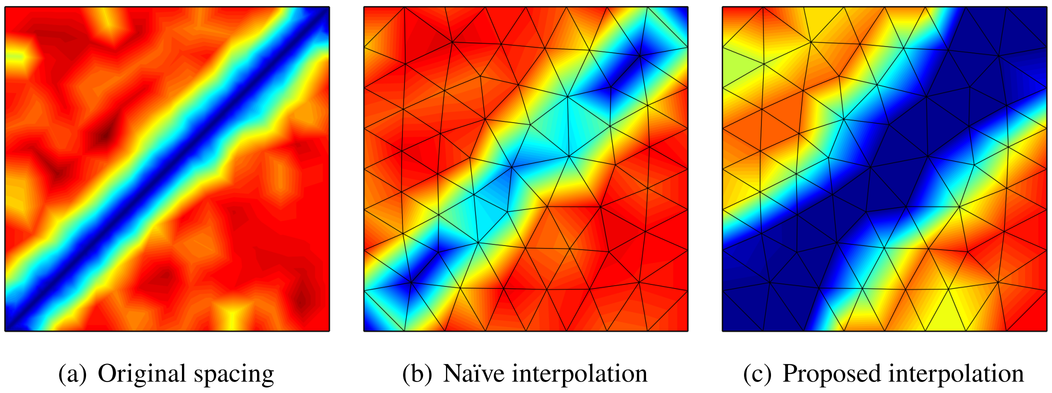
To illustrate the proposed strategy, let us consider the scenario of Figure 6.3. The extract of the mesh with continuous red edges corresponds to the mesh where the solution is available and where the spacing required at the nodes has been computed, as described in Section 6.1. The extract of the mesh with discontinuous black edges corresponds to the background mesh. The objective is to obtain the spacing at the nodes of the background mesh, $\boldsymbol{x}_a$, $\boldsymbol{x}_b$ and $\boldsymbol{x}_c$ in Figure 6.3 so that the spacing at the nodes of the fine mesh can be accurately reproduced.

A naïve interpolation approach would consider the element of the fine mesh containing each node of the coarse mesh and perform a linear interpolation of the nodal values. However, this will make very limited use of the rich information available in the fine mesh. If values of the spacing are interpolated in this way, it is possible to obtain very large values of the spacing at the nodes of the background mesh even if very small values are present in the vicinity of the nodes of the background mesh. Referring to the example of Figure 6.3, if for instance the spacing is very small at nodes $\boldsymbol{x}_6$, $\boldsymbol{x}_7$, $\boldsymbol{x}_8$ and $\boldsymbol{x}_9$, but it is very large at the remaining nodes, a naïve interpolation will compute a large value of the spacing at the nodes $\boldsymbol{x}_a$, $\boldsymbol{x}_b$ and $\boldsymbol{x}_c$ of the background mesh. This will induce a spacing function not suitable to capture the initial solution.

To avoid this problem, a different strategy is proposed to interpolate the spacing. For each element of the background mesh, the list of nodes of the fine mesh that are contained in the background element is identified. In the example of Figure 6.3 all the numbered nodes, from $\boldsymbol{x}_1$ to $\boldsymbol{x}_{13}$ are identified. A very conservative approach is then adopted in this work, which is to define the spacing at the element nodes of the background mesh as the minimum of the spacing of all the nodes of the fine mesh contained in the element. This strategy will ensure that the resulting spacing is certainly able to capture the required solution. Other strategies that can be explored include the use of the arithmetic mean, the harmonic mean, or a weighted arithmetic mean.

The process is finalised by assigning to each node of the background mesh the minimum of the spacings computed from each element sharing this node.

An example is shown in Figure 6.4 to illustrate the process. The spacing function obtained on a reference mesh is transferred to a background mesh using a naïve interpolation approach and the proposed approach. It can be clearly observed that the naïve approach does not produce an accurate representation of the original spacing function.



(a) Original spacing (b) Naïve interpolation (c) Proposed interpolation

Figure 6.4: Illustrative example of two possible interpolations of the spacing onto a background mesh.

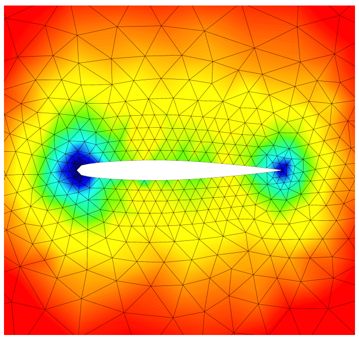


Figure 6.5: Spacing function on a background mesh after interpolating the spacing of Figure 6.2(b).

When used for mesh generation, this background spacing will lead to a mesh that is not capable of representing the features of the target solution. In contrast, with the proposed interpolation, a conservative approach is favoured and the resulting spacing will lead to a finer mesh, ensuring that all the features of the target solution are captured when a mesh is generated with this spacing.

Figure 6.5 shows the interpolated spacing on a background mesh for the example of Figure 6.2(b). The example clearly illustrates the ability of the proposed interpolation strategy to capture the required spacing on a coarse background mesh. It is worth noting that the proposed approach is designed to produce a spacing capable of capturing all the required solution features. However, when the background mesh is excessively coarse, it can produce a spacing function that leads to an over-refined mesh.

6.4 Using a neural network to predict the spacing

The two strategies presented above are designed to preprocess a dataset of available solutions and produce a dataset suitable for training a neural network. The inputs of the neural network are design parameters (e.g., boundary conditions, geometry). The examples considered in this work involve inviscid compressible flows in three dimensions and the design parameters are the flow conditions, namely the free-stream Mach number and the angle of attack.

For the strategy based on sources, the output consists of the position (three coordinates), the spacing and the radius of the global set of sources. For the second approach, based on a background mesh, the output is simply the spacing at the nodes of the background mesh. It is worth noting that the use of a background mesh implies a reduction of the number of outputs by a factor of five, when compared to the strategy based on sources.

In general, the values of the spacing, in both approaches, and the radius, in the first approach, vary by more than two orders of magnitude. To facilitate the training of the NN, these outputs are scaled logarithmically. The scaling not only prevents a bias towards larger values but also prevents the prediction of unrealistic negative values.

The type of NN employed in this work is a standard multi-layer perceptron and extensively described in the literature (Hagan et al., 1997; Balla et al., 2021). In terms of the implementation, TensorFlow 2.7.0 (Abadi et al., 2016) is employed to construct the NNs. To minimise the influence of the random initialisation of the weights, each training is performed five times by performing a variation of the initial guess used in the optimisation. The maximum number of iterations allowed for the optimisation is 500, and the process is stopped either when this number of iterations is reached or when the objective cost function does not decrease during 50 consecutive iterations.

Through preliminary numerical experimentation on the influence of the activation function on the accuracy of the NN – outlined in Appendix C, the sigmoid activation function was chosen as it tended to produce more accurate results compared to other classical activation functions. Therefore, for each NN produced, the sigmoid function was employed for all the hidden layers, with a linear function being used on the output layer. Respectively, these activation functions are given by

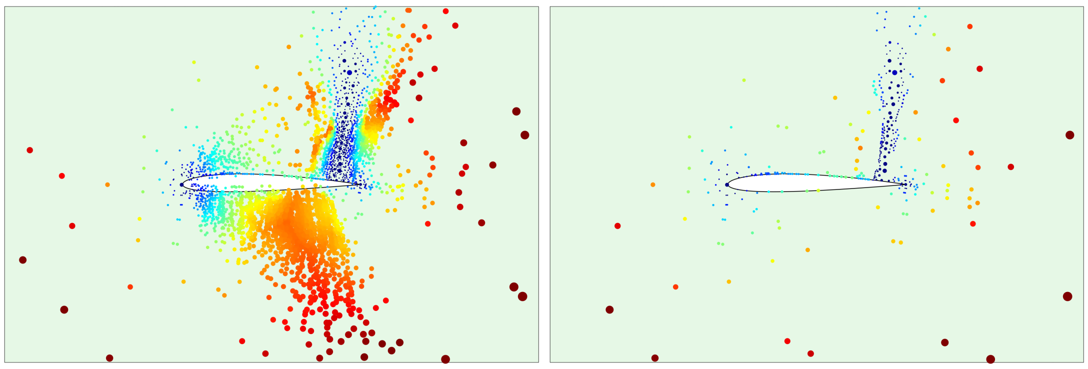
$$S(x) = \frac{1}{1+e^{-x}} \qquad \text{and} \qquad L(x) = x. \tag{6.1}$$

To train the NNs, the cost function used is the mean square error (MSE), with the optimisation function used to minimise the cost being the ADAM optimiser (Kingma and Ba, 2014), with a learning rate of 0.001.

As usual in this context, the hyperparameters of the NN are tuned to ensure that the best architecture is employed. As in Chapter5, it was demonstrated that even performing a fine tuning of the hyperparameters and repeating the training five times, the resulting approach is more efficient than the usual practice in industry of generating an overrefined mesh to perform the simulations for varying flow conditions using a fixed grid.

The design of the NNs considered in this work requires selecting an appropriate number of layers, number of neurons per layer and activation functions. For each numerical example in Section 6.5, the number of layers N_l and the number of neurons in each layer N_n is varied in the pursuit of finding the optimal hyperparameter configuration. The hyperparameter variation is defined by a grid using the ranges $N_l = [1, 2, \dots, 5, 6]$ and $N_n = [25, 50, \dots, 225, 250]$.

The accuracy of the predictions is measured using the statistical R^2 measure (Glantz and Slinker, 2001). To better analyse the results, when the approach using sources is considered, the R^2 measure is reported independently for the five source characteristics (i.e., the three coordinates of the source, the spacing and the radius).



(a) Predicted sources (b) Reduced sources

Figure 6.6: Predicted global sources for an unseen case and the resulting sources after removing redundant sources.

6.4.1 Spacing prediction using sources

After the NN is trained, it is used to predict the characteristics of the global sources for cases not seen during the training stage. It is possible to directly use the predicted global sources to define the mesh spacing function that is required to generate a near-optimal mesh. However, due to the use of a global set of sources, it is expected that the predicted sources for a new case contain redundant information. For this reason, an extra step is required with this technique to minimise the number of queries that the mesh generation requires to define the spacing at a given point. The process, described in detail in Section 5.1.4, involves removing sources with an associated spacing function that can be described by other sources. In addition, an attempt is made to reduce the number of sources by merging point sources into line sources when possible.

Figure 6.6 shows the result of the process used to reduce sources for a predicted global set of sources.

6.4.2 Spacing prediction using a background mesh

In this case, once the NN is trained, it can be used to predict the spacing at the nodes of the background mesh. With this approach, there is no need to perform any further processing of the predicted data, and it can be directly used by a mesh generator to obtain the near-optimal mesh for an unseen case.

6.4.3 Background Mesh Sizing and its Influence on Learning Performance

The performance of the proposed method is closely linked to the structure and resolution of the background mesh used to represent the spacing field. Since the neural network is trained to predict the spacing values at the nodes of a fixed background mesh, the number and distribution of these nodes directly define the size and nature of the learning task.

The method outlined in the chapter exhibits a degree of robustness to background mesh sizing, owing to the use of a conservative interpolation strategy when transferring spacing fields from the CFD mesh – Section 6.3.1. This ensures that key flow features are not lost, even when the background mesh is relatively coarse. However, mesh resolution remains an important factor. If the background mesh is too coarse, the patches over which spacing information is aggregated become overly large, leading to broad refinement zones that can introduce excessive and unnecessary mesh density in downstream generation. Conversely, if the background mesh is too fine, the number of output nodes – and therefore the number of predicted values, rises substantially. This increases the complexity of the learning task and may degrade accuracy due to limited training data. In practice, this effect was observed in (Sanchez-Gamero et al., 2024), where models trained with finer background meshes achieved lower R^2 scores, indicating diminished generalisation performance.

To assess the sensitivity of model performance to background mesh resolution, a range of background meshes were experimented with, varying in size across several orders of magnitude (approximately 1k, 10k, and 100k nodes). This manual variation allowed for the identification of a suitable mesh density that balances learning complexity with spatial resolution. Although a fully automated procedure for background mesh sizing remains an avenue for future work, the results presented in this chapter use meshes that were selected through this targeted exploration and found to offer strong performance across both isotropic and anisotropic scenarios.

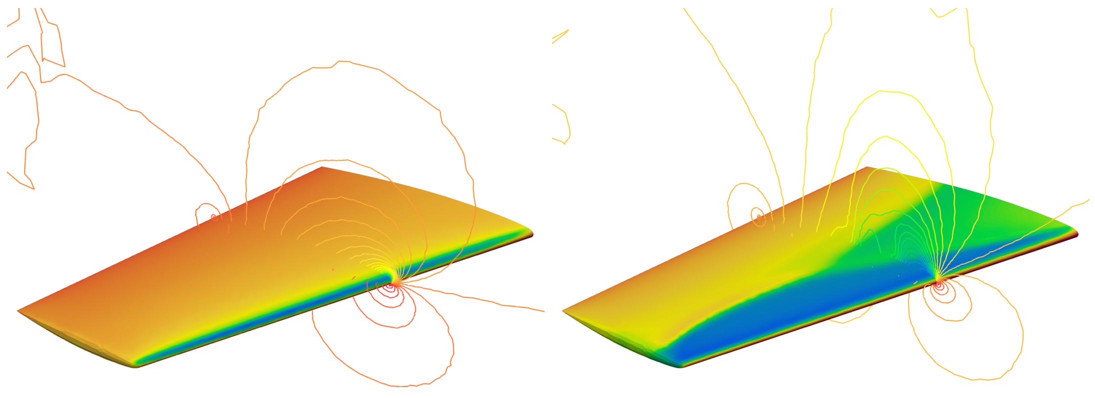
6.5 Numerical examples

This section presents a numerical example to demonstrate the potential of both approaches and to compare their performance. The example involves the prediction of near-optimal meshes for three-dimensional inviscid compressible flow simulations over a wing for varying flow conditions. A second numerical example is presented to show the ability of the approach best performing for the first example in a more realistic scenario involving the inviscid compressible flow around a full aircraft configuration. In both examples the variation of the flow conditions considered induce a significant variation of the solution and s subsonic and transonic flows. All the CFD simulations used in this work were conducted using the in-house flow solver FLITE (Sørensen et al., 2003a).

6.5.1 Near-optimal mesh predictions on the ONERA M6 wing

The ONERA M6 wing (Schmitt, 1979) is considered for this example and flow conditions are described by the free-stream Mach number, M_∞, and the angle of attack, α. The range used for the two design parameters, $M_\infty \in [0.3, 0.9]$ and $\alpha \in [0°, 12°]$, leads to subsonic and transonic flows. Therefore, the mesh requirements for different cases are substantially different, posing a challenge in the prediction of the near-optimal mesh for a given set of parameters.

The variation of the solution that is induced by the variation of the parameters is illustrated in Figure 6.7, showing the pressure coefficient, C_p, for two flow conditions. For the subsonic case, with $M_\infty = 0.41$ and $\alpha = 8.90°$, the solution requires refinement



(a) $M_\infty = 0.41, \alpha = 8.90^\circ$ (b) $M_\infty = 0.79, \alpha = 5.39^\circ$

Figure 6.7: Pressure coefficient, C_p, for the ONERA M6 wing and for two flow conditions.

only near the leading and trailing edges. In contrast, for the transonic case, with $M_\infty = 0.79$ and $\alpha = 5.39^\circ$, the mesh should be refined to also capture the λ-shock on the top surface. The simulations were conducted using tetrahedral meshes with approximately 1.3M elements and 230K nodes.

For the purpose of this study, training and testing data sets were generated by employing a Halton sampling (Halton, 1964) in the parametric space. The training set comprises $N_{tr} = 160$ cases, whereas the test set is made of $N_{tst} = 90$ cases. To ensure that the conclusions are not biased by an incorrect use of the NN for extrapolation, the range of values used to generate the test set is slightly smaller than the range used to generate the training data.

The approach using sources, required between 2,142 and 5,593 sources to represent the spacing of each training case. When combined, the resulting number of global sources is 19,345. This means that the number of outputs of the NN to be trained is almost 100K. For the second approach a background mesh with 14,179 nodes is employed, meaning that the NN to be trained has almost seven times less outputs when compared to the approach that uses sources.

After tuning the NN that best predicts the spacing, both approaches can be compared. For one of the 90 unseen cases Figure 6.8 shows the regression plot for the spacing

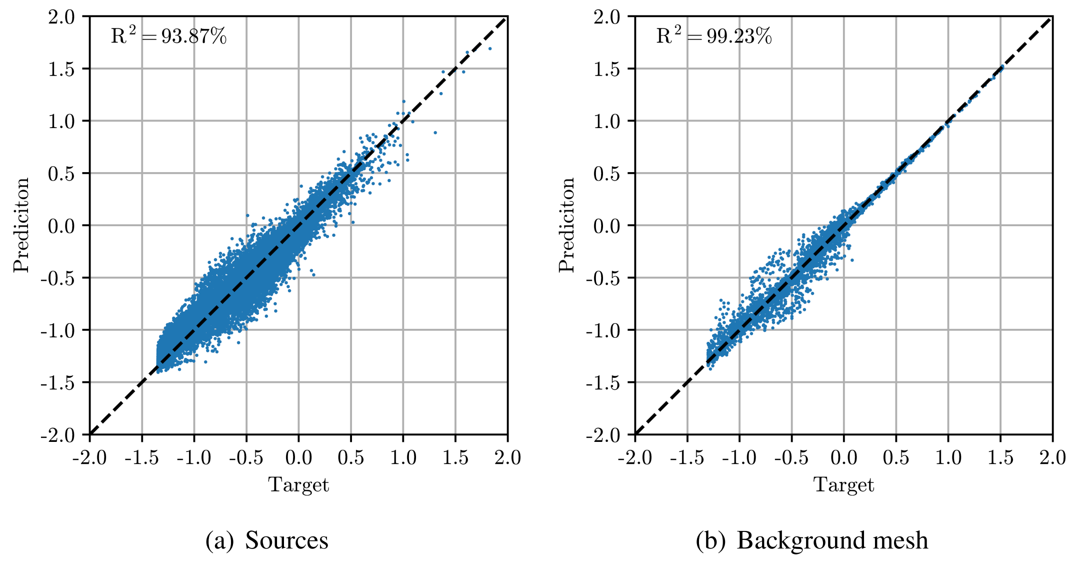


Figure 6.8: The regression plots for the spacing, δ_0, for the approach using sources and the approach using a background mesh.

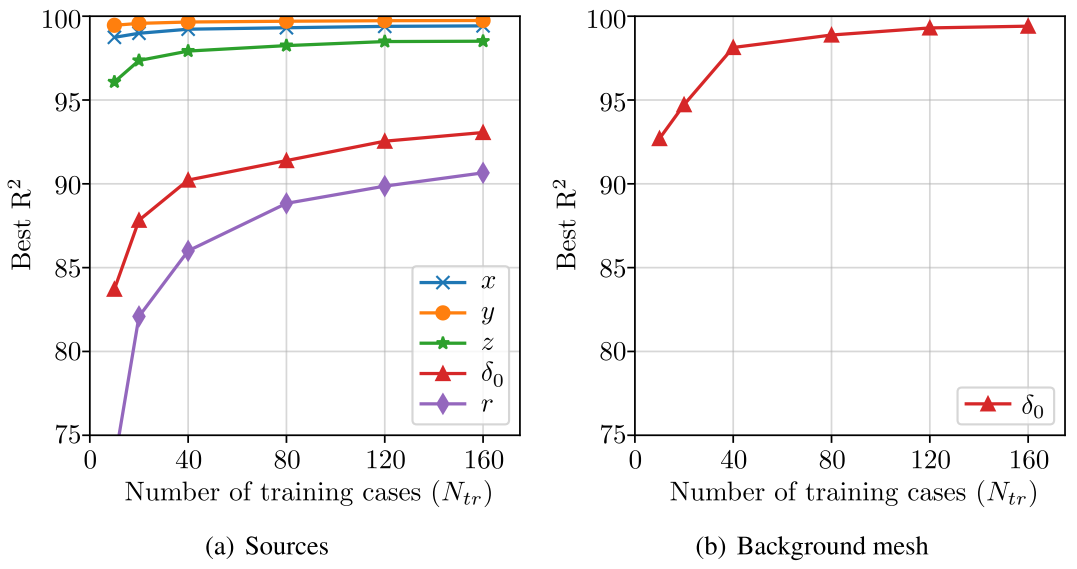


Figure 6.9: ONERA M6: Minimum R^2 for the predicted outputs as a function of the number of training cases for the two methods.

for both approaches. The results indicate a better performance of the approach using a background mesh for this particular unseen case.

To better compare the accuracy, the minimum R^2 for each of the 90 unseen test cases is taken and compared in Figure 6.9, for an increasing number of training cases. The results show that the strategy that uses sources leads to a very accurate prediction of the location of the sources. However, predicting the spacing and the radius of influence

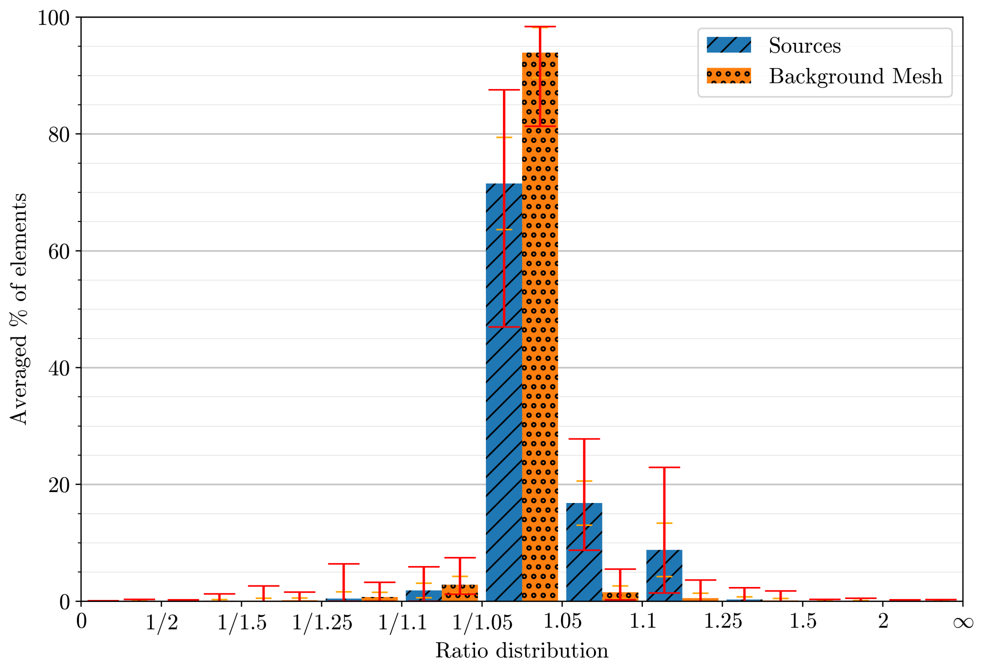


Figure 6.10: ONERA M6: Histogram of the ratio between the predicted and target spacing for the two strategies.

seems much more challenging. To achieve an R^2 of 90 in all the outputs, the whole training data set, with 160 cases, must be considered. In contrast, for the strategy that uses a background mesh 10 training cases are enough to provide an R^2 above 90. By comparing the results, it is clear that the approach that uses a background mesh is significantly more efficient as with 10 training cases the results are as accurate as with the approach that uses sources employing 160 training cases. It is also worth remarking that the approach that uses sources requires the training of multiple NN, whereas only one NN is to be trained by the approach proposed here. In this example the tuning and training of the NN for the proposed approach is almost four times faster than the approach using sources.

To further analyse the performance of the two approaches, the predicted spacing function through the domain is compared against the target spacing function for the two methods. At the centroid of each element of a target mesh, and for all test cases, the spacing induced by the two strategies is compared to the target spacing. Figure 6.10 shows a histogram of the ratio between the predicted and target spacing for both methods. The results correspond to both approaches using all the available training data.

Red bars are used to depict the minimum and maximum values for each bin in the histogram and the standard deviation from the mean is represented by the orange bars. A value of the ratio of spacings between 1/1.05 and 1.05 is considered accurate enough to generate a mesh that is capable of resolving all the required flow features. Values higher than 1.05 where the predicted spacing is larger than the target spacing and, analogously, values below 1/1.05 indicate regions where the NN prediction will induce more refinement than required.

The results in Figure 6.10 clearly illustrate the superiority of the strategy proposed in this work, by using a background mesh. The strategy based on sources provides approximately 70% of the elements with an appropriate spacing whereas the approach based on a background grid accurately predicts the spacing for almost 95% of the elements. In addition, the worst performing case for the approach using sources is less accurate than the worst case for the approach that uses a background mesh. Finally, it is worth mentioning that when the background mesh approach is less accurate, it tends to produce a smaller spacing, which is preferred to a larger spacing, as this will ensure that all solution features are resolved with the predicted near-optimal mesh. This tendency to over-refine can be explained by the conservative interpolation scheme that has been introduced in Section 6.3.1.

Given the high accuracy observed in Figure 6.9 for the approach that uses a background mesh with very few training case, Figure 6.11 shows the histogram of the ratio of between predicted and target spacing for an increasing number of training cases. The results corroborate the conclusions obtained from the R^2 measure in Figure 6.9 and show that with a significantly smaller number of training cases, the approach using a background mesh not only produces an R^2 comparable to the approach with sources with all training data, but also the predicted spacing is as accurate.

To illustrate the potential of the strategies being compared, the trained NNs are used to predict the spacing function for unseen cases and near-optimal meshes are generated and compared to the target meshes. It is worth remarking that the approach that uses

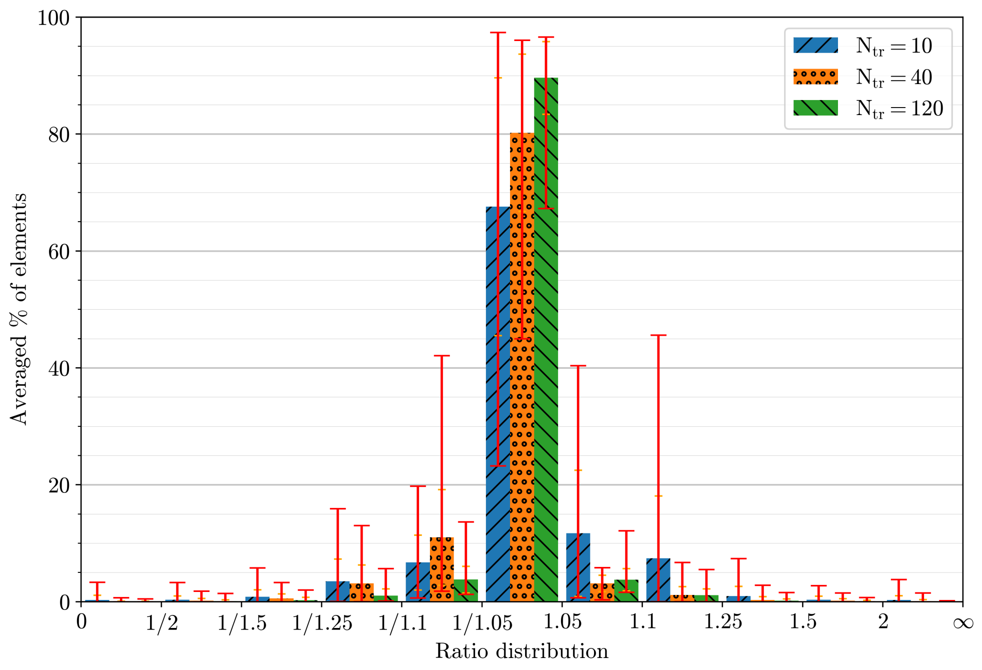


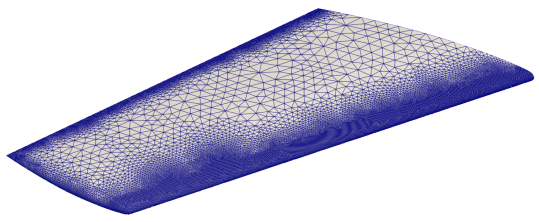
Figure 6.11: ONERA M6: Histogram of the ratio between the predicted and target spacing for the strategy using a background mesh for an increasing number of training cases.

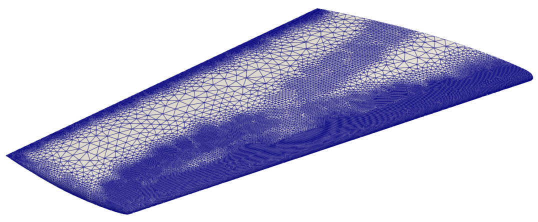
sources undergoes the extra processing step to reduce and merge sources as mentioned in Section 6.4.1.

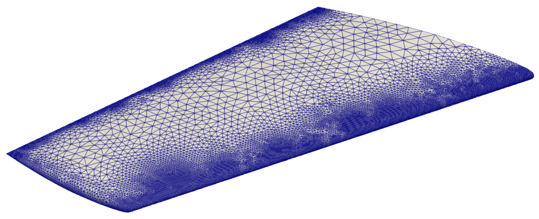
Figure 6.12 shows two target meshes and the near-optimal mesh prediction obtained with the strategy based on sources for two test cases not seen during the training of the NN. The comparison between target and predicted meshes using the strategy based on a background mesh is shown in Figure 6.13 It is worth noting that the target meshes for the two strategies considered are slightly different due to the different definition of the target spacing function.

The results visually show superior accuracy of the proposed approach, based on a background mesh. Not only the meshes obtained with the predicted spacing functions resemble the target more than the meshes predicted with sources, but also the spacing gradation is visually smoother with the approach based on a background mesh.

Further numerical experiments, not reported here for brevity demonstrate that the CFD calculations on the near-optimal predicted meshes result in accurate CFD simulations.


(a) $M_\infty = 0.41, \alpha = 8.90°$


(b) $M_\infty = 0.79, \alpha = 5.39°$


(c) $M_\infty = 0.41, \alpha = 8.90°$

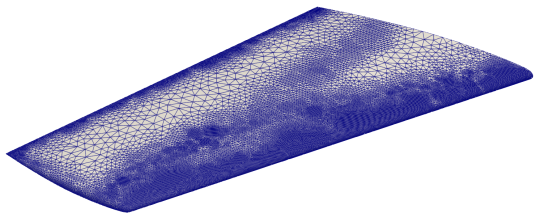
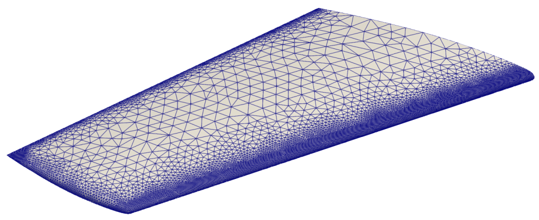
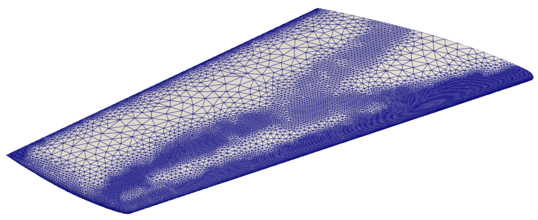
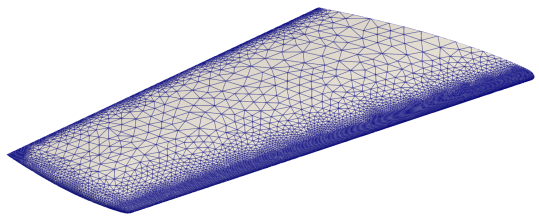

(d) $M_\infty = 0.79, \alpha = 5.39°$

Figure 6.12: Target (top row) and predicted (bottom row) meshes using the strategy based on sources.


(a) $M_\infty = 0.41, \alpha = 8.90°$


(b) $M_\infty = 0.79, \alpha = 5.39°$


(c) $M_\infty = 0.41, \alpha = 8.90°$

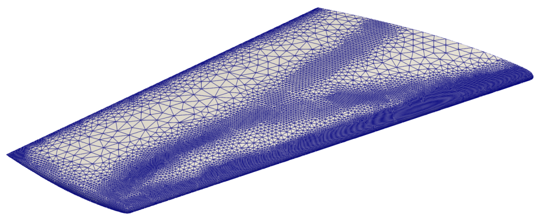

(d) $M_\infty = 0.79, \alpha = 5.39°$

Figure 6.13: Target (top row) and predicted (bottom row) meshes using the strategy based on a background mesh.

More precisely, the aerodynamic quantities of interest (e.g., lift and drag) are obtained with the required accuracy for the aerospace industry.

6.5.2 Near-optimal mesh predictions on the Falcon aircraft

After demonstrating the superiority of the approach based on a background mesh, this section considers an example with a more complex and realistic geometry to show the

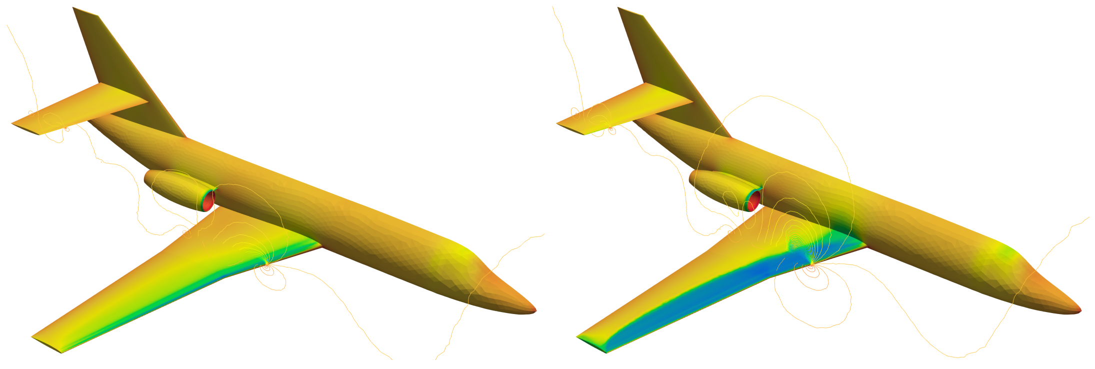

(a) $M_\infty = 0.41, \alpha = 4.50°$ (b) $M_\infty = 0.71, \alpha = 8.00°$

Figure 6.14: Falcon aircraft: Pressure coefficient, C_p, for two different flow conditions.

potential of this approach.

Halton sequencing of the two input parameters is used to generate a training dataset consisting of $N_{tr} = 56$ training cases and $N_{tst} = 14$ testing cases. The range used for the parameters is $M_\infty \in [0.35, 0.8]$ and $\alpha \in [-4°, 10°]$, leading, again, to subsonic and transonic flow regimes.

For each training and test case, the CFD solution is obtained using FLITE (Sørensen et al., 2003a) on an unstructured tetrahedral mesh consisting of 6M elements and 1M nodes. The distribution of the pressure coefficient for two test cases is shown in Figure 6.14. The Figure shows the different flow features that are induced by a change in the design parameters.

To represent the spacing function, the spacing is first determined at each node of the mesh where the solution was computed. A coarse unstructured background mesh is then generated, containing approximately 30K tetrahedral elements, and the spacing is interpolated to the background mesh using the technique described in Section 6.3.1.

A NN is then trained and the hyperparameters are tuned, following the procedure described in the previous example. After the training is performed, the spacing is predicted for the 14 unseen test cases and the accuracy of the predictions is evaluated using the R^2 measure. Figure 6.15 shows the minimum R^2, as a function of the number of training cases. The results show that, even for this more complex example, the

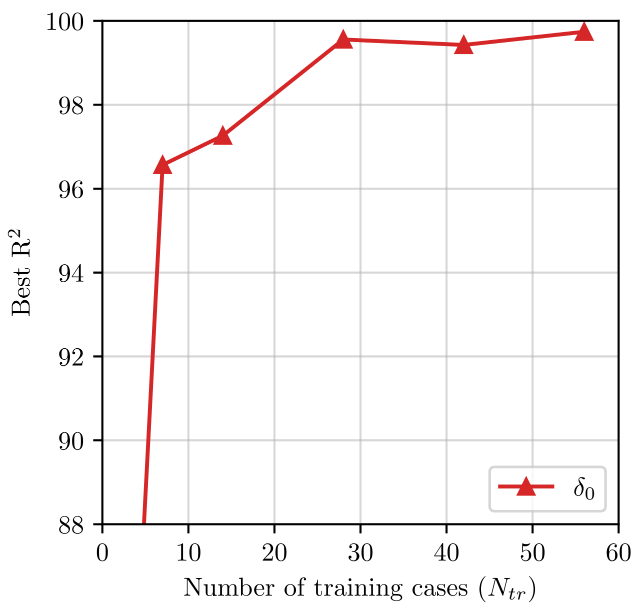


Figure 6.15: Minimum R^2 for the characteristics as a function of the number of training cases.

behaviour is almost identical to the one observed for the previous geometry. With less than 10 training cases the predicted spacing achieves an excellent accuracy, with the value of R^2 above 96%. If the total set of available training cases is considered, the value of R^2 reaches almost 100.

To further assess the accuracy of the predictions, the ratio between predicted and target spacing is evaluated to quantify the performance of NN in producing new meshes for unseen flight conditions. Figure 6.16 shows the histogram of the ratio between predicted and target spacing at the nodes of the background mesh. The minimum and maximum values for each bin in the histogram are depicted with red error bars, whereas the orange bar represents the standard deviation from the mean. A value of the ratio between 1/1.05 and 1.05 is considered sufficiently accurate to produce a mesh able to capture the targeted flow features. The histogram confirms the accuracy of the predictions, with the middle bin containing more than 90% of the elements.

The trained NNs are next used to predict the spacing for the background mesh, from which its subsequent near-optimal mesh is generated and compared with the corresponding target meshes. Figure 6.17 displays the target and ML-produced meshes for

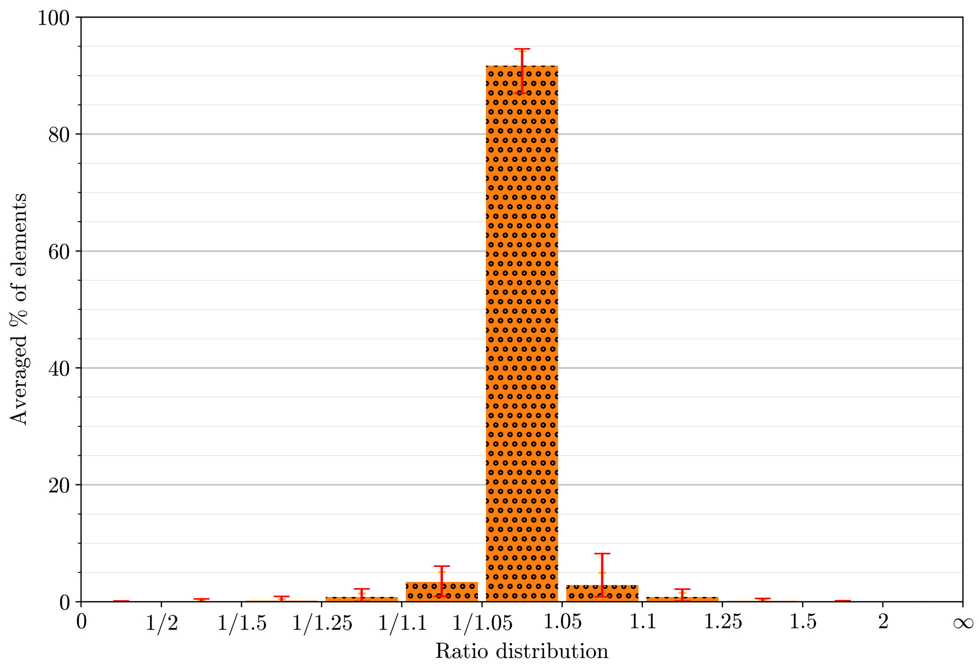
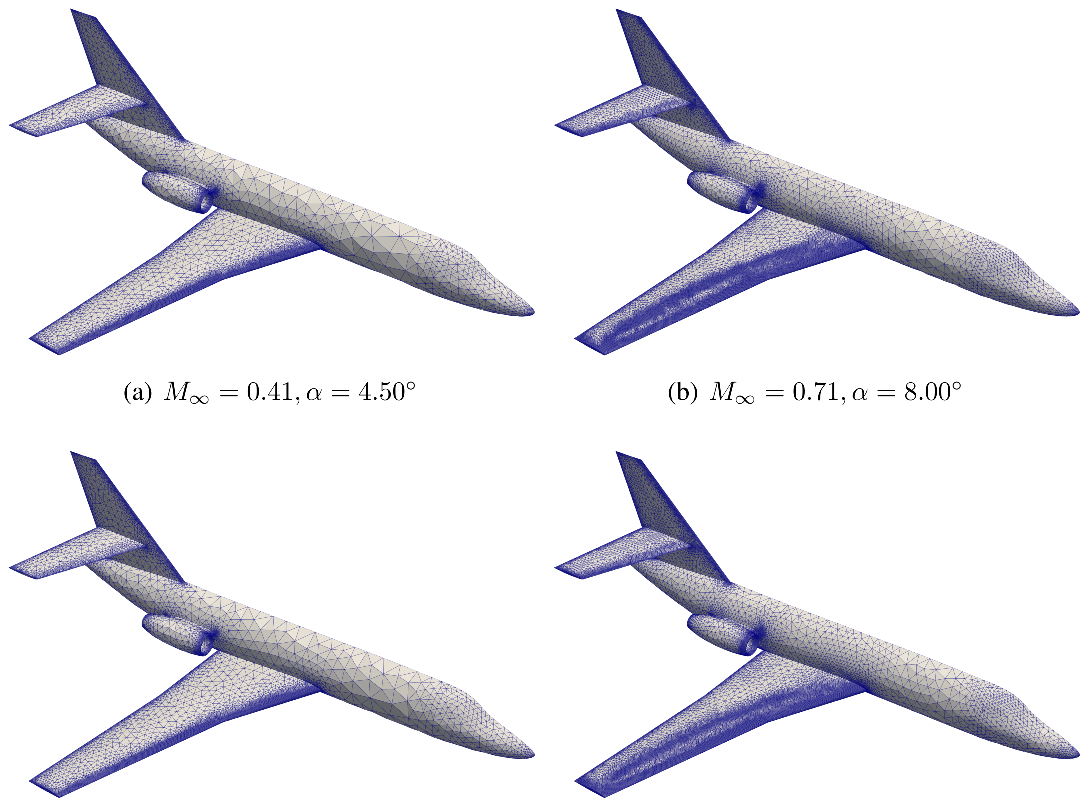


Figure 6.16: Falcon aircraft: Histogram of the ratio between the predicted and target spacing.

the two unseen examples outline in Figure 6.14. The results clearly show the ability of the proposed technique, based on a background mesh, to automatically produce meshes that are locally refined near the relevant regions. For the subsonic case, the NN has appropriately refined the leading and trailing edges of the main wing, the vertical and horizontal stabiliser, as well as the entry and exit of the jet engine. Similarly, for the transonic case, those features are also appropriately captured, but in addition, the NN has successfully predicted the presence and location of a shock along the main wing and consequently appropriately refined this region.

6.6 Concluding remarks

This chapter has demonstrated that both the source-based and background-mesh-based approaches are capable of accurately predicting spacing fields suitable for generating near-optimal meshes. Whilst the source method offers a compact representation and



(a) $M_\infty = 0.41, \alpha = 4.50°$
(b) $M_\infty = 0.71, \alpha = 8.00°$
(c) $M_\infty = 0.41, \alpha = 4.50°$
(d) $M_\infty = 0.71, \alpha = 8.00°$

Figure 6.17: Falcon aircraft: Target (top row) and predicted (bottom row) meshes for two flow conditions.

exhibits flexibility for handling geometric variation, the background mesh approach provides notable advantages in terms of implementation simplicity, learning stability, and spatial consistency.

In particular, the background mesh formulation proved to be highly effective. By transferring the target spacing field onto a fixed reference mesh and training the neural network to predict spacing values at its nodes, the method enabled accurate and generalisable mesh prediction using a relatively modest dataset. The fixed structure of the background mesh also allows the neural network architecture to remain consistent across all cases, streamlining the training process.

A further advantage of the background mesh approach lies in its natural compatibility with anisotropic mesh specification. As will be discussed in the next chapter, this structure readily accommodates anisotropic metric tensors, allowing directional refinement to be captured and predicted more directly than in the source-based method. This provides a path towards generating high-quality meshes tailored not only in size but also in directional resolution, which is critical for resolving flow-aligned features in complex geometries.

Chapter 7

Predicting the Near-Optimal Anisotropic Mesh Using a Background Mesh

This chapter presents a machine learning-based strategy for predicting anisotropic mesh spacing fields using a background mesh representation. Building on the methodology introduced in Chapter 6, the approach is extended to predict full metric tensors rather than scalar spacing values, enabling the generation of anisotropic meshes. In addition, the method is adapted to handle varying geometries, allowing it to produce near-optimal meshes for unseen simulations with different geometric configurations or flow conditions.

The workflow comprises three main stages. First, the metric tensor is computed at each node of a high-fidelity mesh using Hessian-based error analysis, capturing the local anisotropic spacing required to resolve key flow features. Second, a conservative interpolation technique transfers the computed metric field to a common coarse background mesh. This step includes a mesh morphing procedure to accommodate geometric variability across cases. Finally, a neural network is trained to predict the components of the metric tensor directly from flow and geometric parameters.

Several neural network configurations are explored and compared, including different strategies for predicting anisotropic directions and corresponding spacings. The proposed methodology is evaluated using two numerical examples involving three-dimensional inviscid compressible flows. The first focuses on the ONERA M6 wing under varying flow conditions; the second considers a full aircraft geometry defined by eleven design parameters. In both cases, the predicted meshes are assessed in terms of their ability to capture key flow features and produce accurate aerodynamic predictions. The performance of the proposed models is quantified and discussed, demonstrating the potential of this approach to support efficient and automated mesh generation in complex aerodynamic simulations.

7.1 Near-optimal anisotropic mesh spacing prediction

This work presents a new approach to predict the anisotropic spacing function on a background mesh that can be used to generate meshes suitable for unseen simulations that involve either new geometric configurations or new flow conditions. It is assumed that the data for training is available from historical high fidelity analysis.

The proposed methodology can be summarised in the following stages:

1. For each solution that is available as training data, identify the anisotropic spacing that could be used to generate a mesh capable of accurately capturing the given solution.

2. For each available training case, transfer the anisotropic spacing to a coarse background mesh. This step involves the morphing of the background mesh when geometric parameters are considered.

3. Train a NN to predict the metric tensor that defines the anisotropic spacing at each node of the background mesh.

7.1.1 Computation of the target anisotropic spacing in the computational mesh

The construction of a discrete spacing function from a given solution is based on classical error analysis techniques, as detailed in Appendix A. In this work, pressure, p, is selected as the key variable due to its sensitivity to gradients and discontinuities in inviscid compressible flow simulations.

In Chapters 5 and 6, isotropic spacing was employed, where only the largest eigenvalue of the Hessian matrix was considered, leading to uniform refinement in all directions. However, in the anisotropic setting considered here, it is necessary to account for directional variation in the second derivatives of the pressure field. This is achieved by considering the full decomposition of the Hessian matrix at each node, allowing mesh refinement to be tailored independently in the three orthogonal directions.

The spacing at a node $\mathbf{x}_i$ in a given direction, defined by a unit vector $\boldsymbol{\beta}$, is governed by the expression

$$\delta_{i,\boldsymbol{\beta}}^2 \left(\sum_{k,l=1}^{3} (H_i)_{kl} \beta_k \beta_l \right) = K, \tag{7.1}$$

where K is a user-defined constant, and $(H_i)_{kl}$ are the components of the Hessian matrix $\mathbf{H}_i$ at node $\mathbf{x}_i$.

The three mutually orthogonal refinement directions at each node are given by the eigenvectors of $\mathbf{H}_i$, and the corresponding spacings in those directions are computed as:

$$\delta_{i,j} = \begin{cases} \delta_{\min} & \text{if } \lambda_{i,j} > K/\delta_{\min}^2, \\ \delta_{\max} & \text{if } \lambda_{i,j} < K/\delta_{\max}^2, \\ \sqrt{K/\lambda_{i,j}} & \text{otherwise,} \end{cases} \tag{7.2}$$

where $\lambda_{i,j}$ is the j-th eigenvalue of $\mathbf{H}_i$. Equation (7.2) ensures that the computed spacings lie within the user-defined range $[\delta_{\min}, \delta_{\max}]$, avoiding excessive refinement near sharp gradients and overly coarse spacing in smooth regions.

The constant K is defined as

$$K = S^2 \delta_{\min}^2 \lambda_{\max}, \tag{7.3}$$

where $S \in (0, 1]$ is a user-specified scaling factor, taken as 0.2 in all examples, and $\lambda_{\max}$ is the largest eigenvalue of the Hessian at node $\mathbf{x}_i$. Further discussion on the influence of S is available in (Sanchez-Gamero et al., 2024).

Finally, to maintain numerical robustness, the maximum allowable directional stretching is limited to five, based on prior experience that higher anisotropy adversely affects the convergence behaviour of the vertex-centred finite volume solver used in this study.

7.1.2 Transfer of the target anisotropic spacing to a background mesh

Given that the different simulations available as training cases might have been performed in different meshes and a feed-forward NN requires the same number of outputs for all cases, the proposed strategy involves transferring the spacing function of each simulation to the same background mesh. This approach also aims at reducing the number of outputs in the NN, and consequently the training time, as the available simulations might have been performed in extremely fine meshes.

As laid out in Chapter 6, a *conservative interpolation* approach to perform this task when the spacing function is isotropic, which is briefly summarised here. Given a node, $\boldsymbol{x}_i^{\mathtt{B}}$, of the background mesh $\mathcal{B}_h$, the patch of elements that contains the node $\boldsymbol{x}_i^{\mathtt{B}}$ is denoted by $\mathcal{P}_i^{\mathtt{B}}$. The subset of nodes of the computational mesh, corresponding to a training case, that are within the patch $\mathcal{P}_i^{\mathtt{B}}$ is denoted by $\mathcal{X}_{\mathcal{P}_i}$. With this notation, the conservative interpolation of the spacing involves defining the spacing at $\boldsymbol{x}_i^{\mathtt{B}}$ as the minimum of the spacing of all nodes in $\mathcal{X}_{\mathcal{P}_i}$, namely

$$\delta_i^{\mathtt{B}} = \min_{j \in \mathcal{X}_{\mathcal{P}_i}} \{\delta_j\}. \tag{7.4}$$

To transfer the target anisotropic spacing, computed following the strategy described in the previous section, from a computational mesh to a background mesh, this work employs a *conservative metric intersection*. Metric intersection is commonly used to obtain a representative metric when two (or more) metrics are defined at a point. The specific formulation and mathematical details of the intersection process are described in Section 3.2.2.2.

The proposed conservative metric intersection to transfer an anisotropic spacing from a computational mesh to a background mesh is described next. Given a node, $\boldsymbol{x}_i^{\mathtt{B}}$, of the background mesh $\mathcal{B}_h$, the patch of elements that contains the node $\boldsymbol{x}_i^{\mathtt{B}}$, denoted by $\mathcal{P}_i^{\mathtt{B}}$, is considered. The subset of nodes of the computational mesh that are within the patch $\mathcal{P}_i^{\mathtt{B}}$ is denoted by $\mathcal{X}_{\mathcal{P}_i}$. With this notation, the conservative metric intersection involves defining the metric for node $\boldsymbol{x}_i^{\mathtt{B}}$ as

$$\boldsymbol{\mathcal{M}}_i^{\mathtt{B}} = \bigcap_{j \in \mathcal{X}_{\mathcal{P}_i}} \boldsymbol{\mathcal{M}}_j. \tag{7.5}$$

Remark 1. As reported in (Sanchez-Gamero et al., 2024), special attention must be paid to two special cases. First, if the patch of a background mesh node, $\mathcal{P}_i^{\mathtt{B}}$, does not contain any node of the computational mesh, i.e $\mathcal{P}_i^{\mathtt{B}} = \emptyset$, then the proposed strategy is to utilise the metrics of the nodes of the computational mesh that contain the current background mesh node. Second, for domains with curved boundaries, some nodes of the computational mesh might not belong to any patch associated to the background mesh and, consequently, the approach described above would ignore the metrics defined at those points. To avoid this, the nodes of the computational mesh that do not belong to any element of the background mesh are associated to the closest elements of the background mesh. This ensures that the all the metrics available in the computational mesh are utilised when transferring information to the background mesh.

7.1.3 Morphing a Background Mesh

To enable geometric variation across training cases, a consistent background mesh is required so that all spacing fields can be projected onto a common structure suitable for neural network training. However, because the background mesh depends directly on the geometry, each case naturally generates a different mesh, with variations in both the number and arrangement of nodes. This makes it infeasible to directly transfer spacing information from one geometry to another.

To address this, a mesh morphing procedure is introduced. A single background mesh is constructed for a reference geometry, taken as the midpoint of the parametric space. This reference mesh is then deformed to match the geometry of each case in the dataset. The process ensures that all spacing fields are represented on a consistent mesh structure, allowing the neural network to work with fixed-size inputs and outputs. The morphing is performed in two distinct stages: first for the surface, and then for the volume.

In the first stage, the surface mesh is morphed by exploiting the fact that each surface node is defined by parametric coordinates (u_1, u_2) on a smoothly interpolated surface. As the geometry changes, these parametric coordinates remain fixed. The new physical positions of the surface nodes are obtained by reevaluating the surface mapping $\mathbf{r}(u_1, u_2)$ for the updated geometry. This produces a new surface mesh that conforms to the new geometry exactly, without altering the original mesh connectivity.

In the second stage, the interior volume mesh is deformed using barycentric interpolation. A very coarse tetrahedral mesh is first constructed over the reference geometry using only the surface nodes of the background mesh. This defines a set of simple elements that connect the boundary, without introducing additional internal nodes.

For each interior node, the first step is to identify which tetrahedral element of the coarse mesh contains it. Once the enclosing element has been located, the position of the node is expressed as a weighted combination of the four element vertices. These weights, which sum to unity, define the relative position of the node within the element and are referred to as the barycentric coordinates.

As the surface component of the background mesh is deformed to match a new geometry, the associated coarse mesh also deforms, since its nodes lie on the surface. Using the same element connectivity and the previously computed barycentric coordinates for each internal node, the new positions of the nodes are calculated by linear interpolation. This results in a smooth and consistent deformation of the entire background mesh to fit the target geometry. Importantly, this deformation affects only the nodal positions; the connectivity of the background mesh remains unchanged across all geometries.

By performing the surface and volume morphing in this two-stage manner, the reference background mesh is accurately deformed to match each geometry in the dataset. This ensures that all spacing fields are expressed over a shared mesh layout, making them suitable for consistent learning and prediction. Once the background mesh has been morphed to match the geometry of a given case, the new position of each node – both on the surface and in the volume, can be used to identify its surrounding patch of elements, $\mathcal{P}_i^{\mathtt{B}}$. These patches, now defined in the deformed configuration, are then used to collect the relevant spacing or metric values from the computational mesh, as required by the conservative interpolation or metric intersection procedure described in Equation (7.5).

Remark 2. It should be noted that the morphing process described above may result in a background mesh that, by conventional meshing standards, would be considered poor: potentially exhibiting highly skewed or even inverted elements (i.e. elements with negative volume). However, this does not hinder the effectiveness of the proposed approach. Since the morphed background mesh is used solely to define the desired anisotropic spacing for constructing meshes, and not as a computational mesh, its quality is not critical. The downstream mesh generation process remains robust and consistently produces high-quality, flow-suitable meshes based on the prescribed spacing information, regardless of distortions in the morphed background mesh.

7.1.4 NN architecture to predict anisotropic spacing

Using the strategies described in the two previous sections, the tensor metric that defines the target anisotropic spacing in a common background mesh is available, for a set of training cases.

A potential NN architecture would involve setting, as inputs of the NN, the desired parameters (e.g. flow conditions or geometric parameters) and defining, as outputs, the nine components of the metric tensor at all the nodes of the background mesh. However, such a strategy would ignore that a metric tensor must be given by a symmetric definite positive matrix. In addition, predicting the components of the metric tensor directly lacks some interpretability of the predictions as the information about the three mutually orthogonal directions and the spacing in each direction cannot be easily observed by looking at the components of the metric tensor directly.

The orthogonality property of the directions to be predicted can be exploited to reduce the amount of information to be predicted and, at the same time, ensure that the resulting metric tensor satisfies the required properties. To this end, the first direction (corresponding to the minimum spacing) is expressed using spherical coordinates, meaning that only two angles are required, namely α_1 and α_2. Next, to strongly enforce the required orthogonality, the second direction must lie within the orthogonal plane to the first direction, meaning that it can be expressed in polar coordinates in the orthogonal plane using a single angle, namely α_3. Finally, the third direction is uniquely determined from the orthogonality property and there is no need to characterise or predict this direction.

Remark 3. Given the definition of a metric tensor in Equation (3.7), it is easy to verify that the metric tensor given by three spacial directions $\gamma\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$, with spacings δ_1, δ_2 and δ_3, respectively, is identical if $\gamma = 1$ or $\gamma = -1$. Obviously, the same applies if the change of sign is applied to any of the three directions.

The observation in Remark 3 implies that the angles can be restricted to $\alpha_1 \in [0, \pi]$,

Table 7.1: Three models used for training NNs to predict the anisotropic spacing on the background mesh.

Model	NN architecture			
1	NN_1 (δ_1, δ_2, δ_3)	NN_2 (α_1, α_2, α_3)		
2	NN_1 (δ_1, δ_2, δ_3)	NN_2 (α_1, α_2)	NN_3 (α_3)	
3	NN_1 (δ_1, δ_2, δ_3)	NN_2 (α_1)	NN_3 (α_2)	NN_4 (α_3)

$\alpha_2 \in [-\pi/2, \pi/2]$ and $\alpha_3 \in [0, \pi]$. This observation also implies that two dissimilar angles, for instance ϵ and $\pi - \epsilon$, for a small angle ϵ, would produce almost identical metric tensors. It is therefore not advisable to define the angles as the outputs of the NN as a small variation of one anisotropic variation can lead to a large variation of the angle, making the training more difficult and leading to a lower prediction accuracy.

To avoid this issue, each angle is independently mapped to an imaginary circle in the two dimensional plane, namely $\boldsymbol{v}_i = (\cos(2\alpha_i), \sin(2\alpha_i))$, ensuring that two dissimilar angles corresponding to almost identical orientations lead to almost identical vectors $\boldsymbol{v}_i$.

Therefore, the proposed approach consists of building a NN where the inputs are flow conditions or geometric parameters and the outputs are the unit vectors $\boldsymbol{v}_i$ and the corresponding spacings δ_i for $i = 1, \dots, \mathtt{n_{sd}}$.

Three models have been developed and compared in the numerical examples shown later. The first model consists of training two different NNs, one NN to predict the three spacings and a second NN to predict the three vectors $\boldsymbol{v}_i$ that uniquely define the first two directions of anisotropy. The second model consists of training three different NNs, one NN to predict the three spacings, a second NN to predict the two vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ that uniquely define the first direction of anisotropy and a third NN to predict the vector $\boldsymbol{v}_3$ that uniquely defines the second direction of anisotropy. The third model investigated involves training four different NNs, one NN to predict the three spacings and three more NNs to independently predict the vectors $\boldsymbol{v}_1$, $\boldsymbol{v}_2$ and $\boldsymbol{v}_3$. Table 7.1 summarises the three models described.

Each of the models uses a separate loss function tailored to the type of output being predicted. For the scalar spacing values δ_i, a standard mean squared error (MSE) loss is employed, as detailed in Section 4.2.4.1. For the directional vectors $\boldsymbol{v}_i$, which define the anisotropic directions, a cosine similarity loss is used to encourage alignment between the predicted and true unit vectors. The formulation behind this loss is described in Section 4.2.4.2.

In terms of implementation, TensorFlow 2.7.0 (Abadi et al., 2016) is used to construct and train the NNs. To minimise the effect of the random initialisation of the NN weights, each training experiment is repeated five times with different random seeds. In all the examples, each training is performed for a maximum of 5,000 epochs, with early stopping applied if no improvement in the objective function is observed for 100 consecutive epochs. The batch size used in all experiments is eight, which produced better performance compared to the default value of 32 in TensorFlow.

Finally, to evaluate the prediction accuracy of the trained NNs, the classical statistical R^2 measure (Glantz and Slinker, 2001) is used, with results reported separately for each of the predicted outputs.

7.2 Numerical examples

This section presents two numerical examples to assess the accuracy of the proposed strategy to predict the anisotropic spacing for unseen simulations. The first example involves a problem with a fixed wing geometry and variable flow conditions characterised by two parameters. The second example involves a more complex problem with fixed flow conditions and 11 geometric parameters that characterise the shape of a wing in a full aircraft configuration.

All the data used in the examples corresponds to inviscid flow simulations performed with the FLITE system (Sørensen et al., 2003a), a well established vertex-centred finite volume solver.

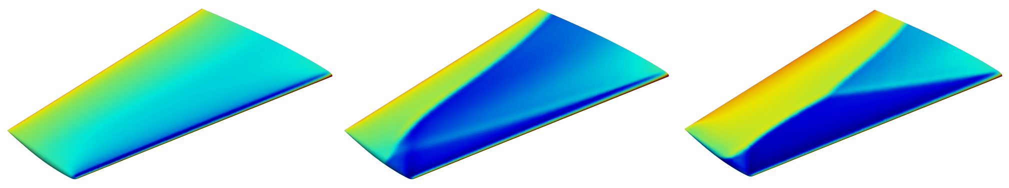

(a) $M_\infty = 0.660, \alpha = 1.91^\circ$ (b) $M_\infty = 0.885, \alpha = 2.13^\circ$ (c) $M_\infty = 0.809, \alpha = 7.12^\circ$

Figure 7.1: ONERA M6 wing: Pressure coefficient, C_p, for three different flow conditions.

7.2.1 Anisotropic spacing predictions on the ONERA M6 wing at various inflow conditions

The first example considers the inviscid compressible flow past the ONERA M6 wing for varying flow conditions, characterised by the free-stream Mach number, M_∞, and the angle of attack α. The variation of the flow conditions is $M_\infty \in [0.6, 0.95]$ and $\alpha \in [0^\circ, 8^\circ]$, encompassing subsonic and transonic flow regimes and leading to a substantial variation of the spacing function required for each simulation.

Figure 7.1 shows three different pressure coefficient distributions over the wing, corresponding to different flow conditions. The first case corresponds to a subsonic flow where the gradients are mainly concentrated along the leading and trailing edges, and anisotropic spacing would be beneficial in these regions. The other two cases involve a transonic flow with a λ-shape shock, but with significant variability of the regions containing steep gradients.

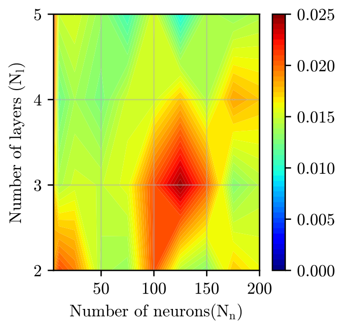
All the simulations available, including training and test cases, were computed on the same tetrahedral mesh with 4.6M elements and 782k nodes and using isotropic spacing. This mesh was selected as the reference (or ground truth) as it produces high-quality solutions across the entire parameter space. It offers smooth pressure distributions, well-resolved leading-edge gradients, and sharp representation of transonic shock structures. As such, it provides a reliable and consistent baseline from which to compute the target metric fields used for training and evaluation.

To select the training and test cases, sampling is performed using Halton sequencing (Halton, 1964). More precisely, scrambled Halton sequencing is employed as it is known to maintain the low-discrepancy in high-dimensional problems (Vandewoestyne and Cools, 2006).

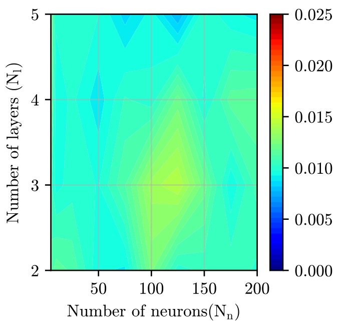
Remark 4. When utilising historical data available in industry, the datasets are unlikely to align with a Halton sequencing. Instead, it is anticipated that the sampling, done by an expert engineer, would be denser in critical regions of the flight envelope where significant changes in flow features are expected to occur. Therefore, although the influence of the sampling method is out of the scope of the current work, it is anticipated that a biased sampling, as done by an expert engineer, would lead to better performance of the trained NN for the same amount of training data or to similar performance with less training data.

For each available case, the metric tensor is computed at each node of the computational mesh using the procedure described in Section 7.1.1. Next, employing the strategy presented in Section 7.1.2, for each training case, the metric is transferred to the same background mesh, which in this example has approximately 590K elements and 100K nodes. Finally, using the procedure described in Section 7.1.4, the spacing and the vectors that describe the first two anisotropic directions are computed, for each case and for each node of the background mesh.

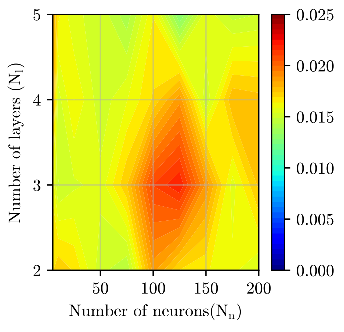
With this information NNs are trained using the parameters described in Section 7.1.4. For the first model, described in Table 7.1, where two NNs are trained separately to predict spacing and anisotropy directions, Figure 7.2 shows the mean average error (MAE) for NNs with an increasing number of hidden layers and neurons, when using $N_{tr} = 40$. The results clearly show that predicting the spacing is significantly easier than predicting the anisotropic directions accurately, with the MAE for spacings being two or three orders of magnitude lower than that for the directions. The number of hidden layers utilised varies from two to five because previous studies involving the prediction of isotropic spacing – in Section 6.5, showed that one hidden layer produces



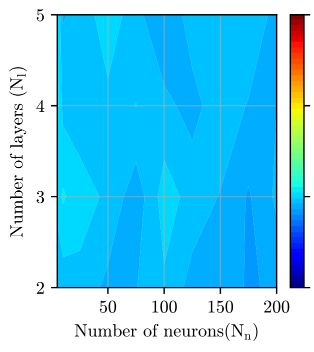
(a) δ_1



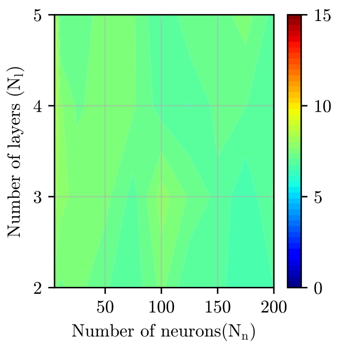
(b) δ_2



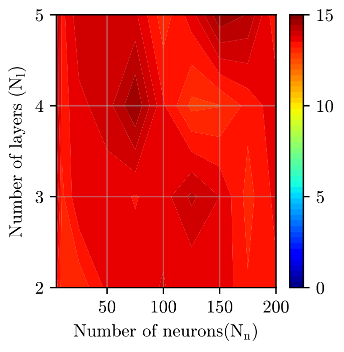
(c) δ_3



(d) α_1



(e) α_2



(f) α_3

Figure 7.2: ONERA M6 wing: MAE for the spacings (δ_i) and angles (α_i) as a function of the number of layers and number of neurons in each layer employing the first model of Table 7.1.

significantly less accurate results and six layers do not bring any benefit in terms of accuracy. In fact, as shown in Figure 7.2, using more than two layers does not provide any extra accuracy. The results also show that the most accurate results are obtained for the first anisotropic direction and its associated spacing. Given that the first direction is associated with the smaller spacing, this is the most important direction to be accurately predicted in order to capture the steep gradients near shocks.

The number of neurons is varied between five and 200, but the results show that using more than 50 neurons does not provide any significant benefit. This means that the training times of the NN are low. For instance using $N_{tr} = 40$ two hidden layers and five neurons per layer, the training of the NN to predict spacing takes approximately seven minutes, whereas the NN to predict the anisotropic directions takes 35 minutes. When the number of neurons is increased to 100 per layer, the training of the NN to predict spacing with takes approximately 23 minutes, whereas the NN to predict the

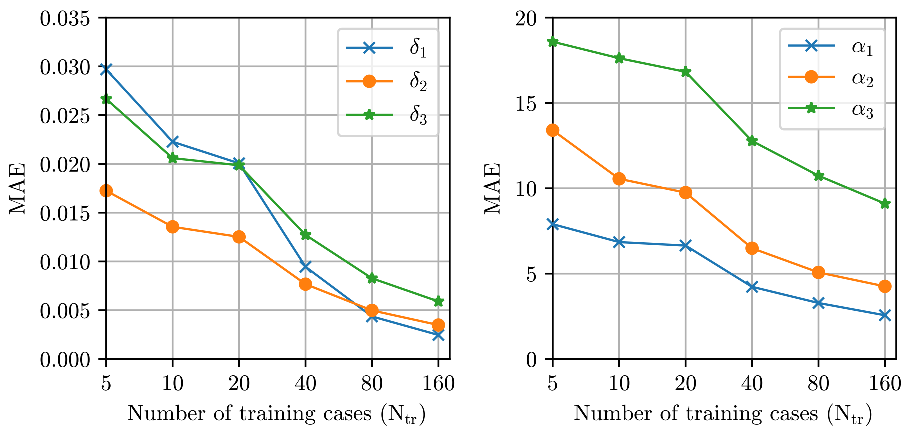
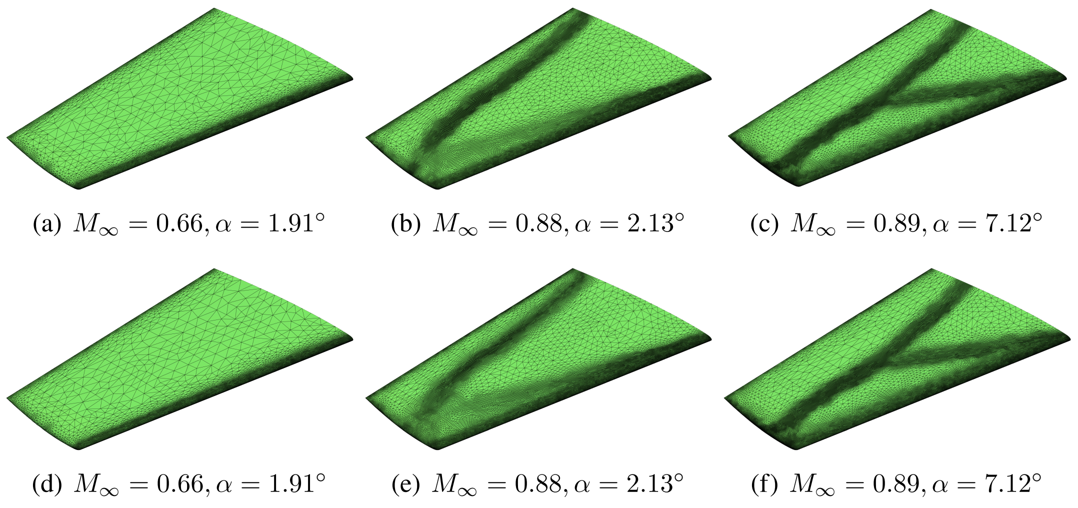


Figure 7.3: ONERA M6 wing: MAE for the spacings (δ_i) and angles (α_i) as a function of the number of training cases, N_{tr}, employing the first model of Table 7.1.

anisotropic directions takes 67 minutes.

Next, the effect of the size of the training dataset is studied. For the first model described in Table 7.1, where two independent NNs are trained, Figure 7.3 shows the maximum MAE for all test cases as a function of the number of training cases for the six predicted outputs. The NN architecture used in this study considers the hyperparameters from the best performing NN, as identified in the hyperparameter tuning process shown in Figure 7.2. The results show a monotonous decrease of the MAE as the number of training cases is increased. For 40 training cases, a significant decrease in the MAE is observed for all the predicted outputs, suggesting that this number of cases provides enough information for the NN to accurately predict the information required to produce suitable anisotropic spacings.

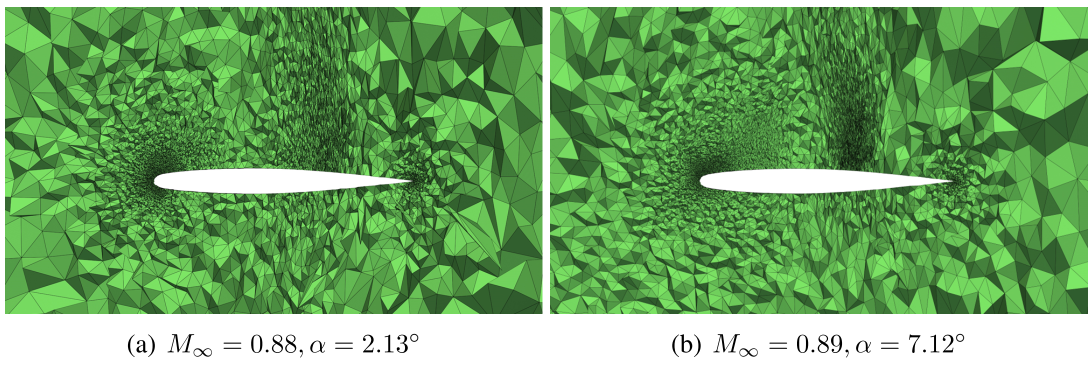
To visually illustrate the potential of the proposed approach, the trained NNs for the first model of Table 7.1 are next used to predict the anisotropic spacing field, which is subsequently used to generate an anisotropic mesh. Figure 7.4 shows the target mesh and the predictions for three inflow conditions corresponding to test cases, unseen by the NN during training. The target meshes are generated by directly using the metric defined in the background mesh. The predicted meshes closely follow the refinement patterns of the target meshes, effectively capturing key features of the flow. For the first case, which is a subsonic test case, the refinement is concentrated in the leading and



(a) $M_\infty = 0.66, \alpha = 1.91^\circ$ (b) $M_\infty = 0.88, \alpha = 2.13^\circ$ (c) $M_\infty = 0.89, \alpha = 7.12^\circ$

(d) $M_\infty = 0.66, \alpha = 1.91^\circ$ (e) $M_\infty = 0.88, \alpha = 2.13^\circ$ (f) $M_\infty = 0.89, \alpha = 7.12^\circ$

Figure 7.4: ONERA M6 wing: Target (top) and predicted (bottom) meshes using the first model for three flow conditions unseen by the NN during training.

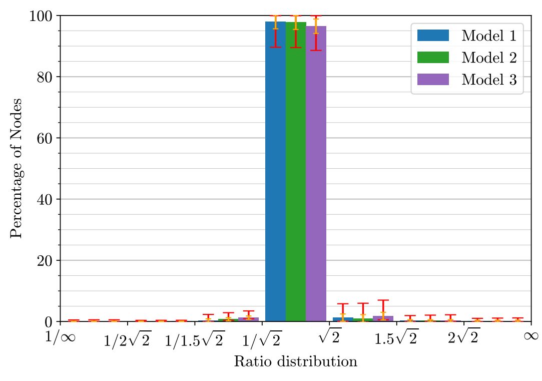


(a) $M_\infty = 0.88, \alpha = 2.13^\circ$ (b) $M_\infty = 0.89, \alpha = 7.12^\circ$

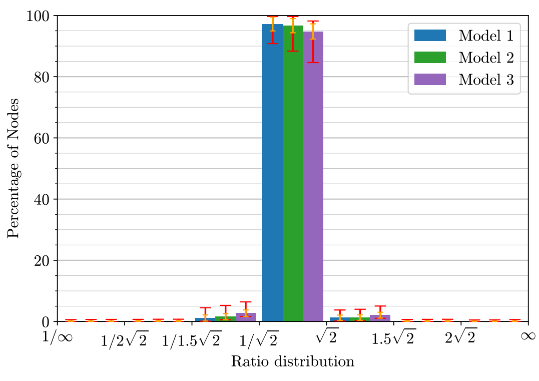
Figure 7.5: ONERA M6 wing: predicted volume meshes using the first model for two transonic flow conditions unseen by the NN during training.

trailing edges and the anisotropy of the predicted spacing matches the one observed in the target mesh. For the two transonic cases, the predicted meshes provide the refinement required to capture the shocks and, again, the anisotropic character of the target spacing is clearly observed in the predicted meshes. The anisotropic spacing is better illustrated for the two transonic cases by showing the volume mesh in Figure 7.5 .

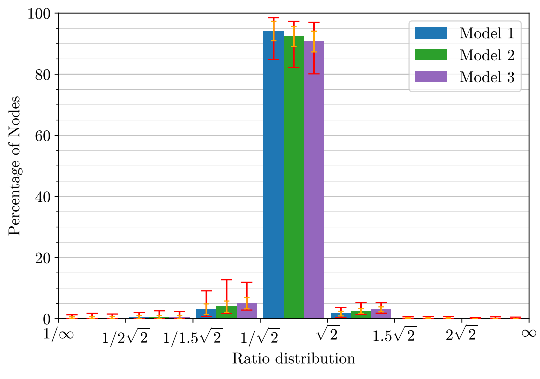
To quantify the performance of the proposed strategy and to compare the different models in Table 7.1, the spacing function predicted by the NN is compared with the target spacing function. It is worth noting that the spacings cannot be directly compared as the target and predicted directions of anisotropy are not identical. Therefore, to



(a) Spacing along $\mathbf{e}_1$



(b) Spacing along $\mathbf{e}_2$



(c) Spacing along $\mathbf{e}_3$

Figure 7.6: ONERA M6 wing: Histogram of the ratio between the predicted and target spacings for the three models of Table 7.1.

produce a suitable error measure, the predicted anisotropic spacings are projected onto the target spacing directions given by $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$ for each mesh node.

Figure 7.6 presents histograms comparing the three models of Table 7.1, evaluated with $N_{tr} = 40$. The histograms show the mean ratio between predicted and target spacing along $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$, across all test cases. The red error bars indicate the minimum and maximum values for each bin, while the orange bars represent the standard deviation from the mean. The bins are defined as multiples of $\sqrt{2}$ and its inverse because, as usual in mesh generation, if a user-specified spacing h at a point is sought, the acceptable values for the spacing are within $h/\sqrt{2}$ and $h\sqrt{2}$. In the histograms, ratios below $1/\sqrt{2}$ indicate more refinement than required, whereas ratios above $\sqrt{2}$ indicate that the mesh is not refined enough.

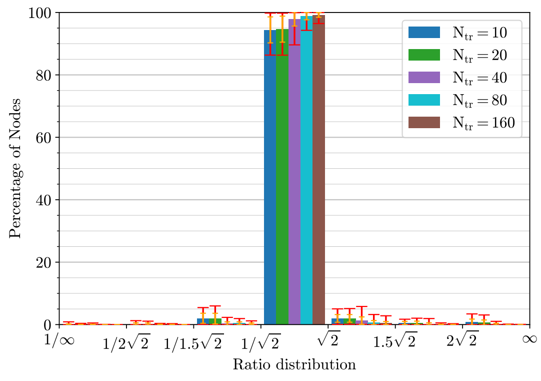
The results show that model 1, with two NNs to predict spacing and directions separately, leads to the most accurate predictions for all directions of the target anisotropic spacing. Slightly less accurate results are obtained for the second model, where the angles associated to the first and second anisotropic directions are predicted by two independent NNs. Finally, the least accurate results are observed for the third model, where each angle is predicted by a different NN. This indicates that using a single NN to predict all the information about anisotropic directions is not only more efficient but also provides the best accurate predictions.

Very small differences between the accuracy of the three models are observed on the predicted spacing along the first target anisotropic direction, $\mathbf{e}_1$, with almost 95% of the mesh nodes having an acceptable prediction with the first model. For the second target anisotropic direction, $\mathbf{e}_2$, the third model shows a sizeable loss of accuracy, whereas the first and second models have almost the same accuracy. The better performance of the first model is best appreciated when considering the spacing along the third target anisotropic direction, $\mathbf{e}_3$. It is also worth mentioning that the first model is the one that exhibits the minimum areas of under-refinement.

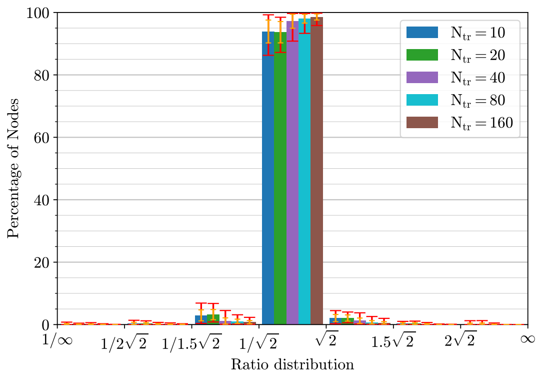
The best performance of the first model is attributed to its ability to leverage global information during training as it is the only model that predicts the two first directions of anisotropy together.

To further quantify the accuracy of the predictions in terms of the training dataset size, Figure 7.7 shows the histogram of the anisotropic spacing accuracy for the first model in terms of the number of training cases, N_{tr}. The histograms show a significant improvement in prediction accuracy for $N_{tr} \geq 40$, as previously shown with the MAE graphs in Figure 7.3. It is worth noting that, for $N_{tr} = 40$ the percentage of nodes with a prediction of the full metric tensor reaches almost 95%. Furthermore, with only 20 training cases, the prediction of the spacing in the first direction (the most critical to accurately represent the shocks) is acceptable for almost 95% of the nodes.

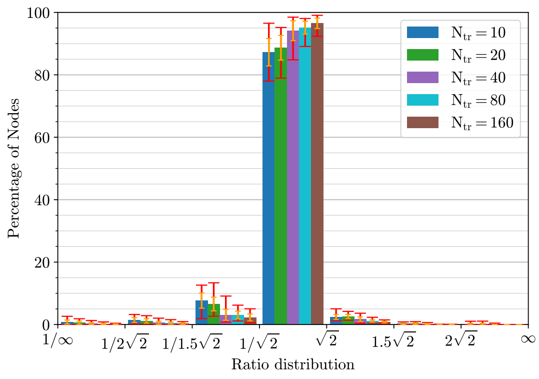
To conclude this example, the suitability of the meshes predicted to perform simulations for unseen cases is studied. To this end, an anisotropic spacing is predicted for three



(a) Spacing in $\mathbf{e}_1$



(b) Spacing in $\mathbf{e}_2$

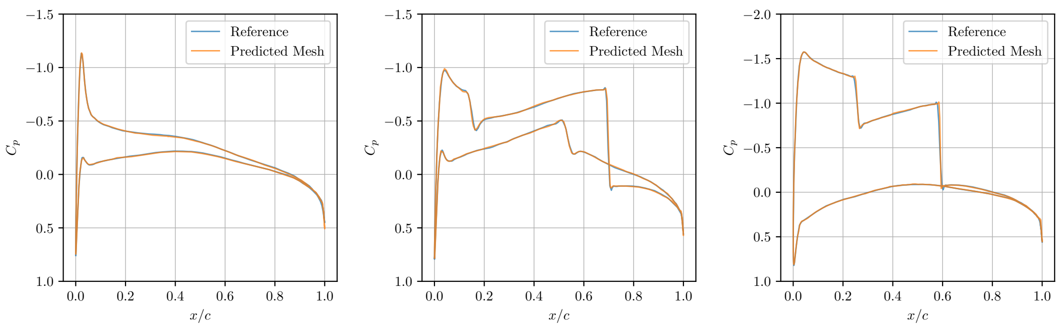


(c) Spacing in $\mathbf{e}_3$

Figure 7.7: ONERA M6 wing: Histogram of the ratio between the predicted and target spacings for the first model of Table 7.1 and for an increasing number of training cases, N_{tr}.

unseen cases, the corresponding anisotropic meshes are generated and simulations are performed. The results, involving pressure coefficient distribution and the lift and drag coefficients, are compared to the results obtained with a fine reference mesh which is produced using mesh adaptivity, starting with the original isotropic mesh with 4.6M elements used to generate all the datasets. Figure 7.8 shows the excellent agreement between the pressure coefficient distributions at one section of the wing for three different flow conditions unseen during the NN training.

To confirm the suitability of the predicted meshes, Table 7.2 compares the reference lift and drag coefficients, C_L and C_D respectively, with the one obtained after performing a simulation with the predicted meshes. The results show a maximum variation in the lift and drag coefficients of only one lift count and two drag counts respectively, clearly demonstrating the suitability of the predicted meshes to perform simulations of unseen



(a) $M_\infty = 0.66, \alpha = 1.91°$ (b) $M_\infty = 0.88, \alpha = 2.13°$ (c) $M_\infty = 0.89, \alpha = 7.12°$

Figure 7.8: ONERA M6 wing: Comparison of the pressure coefficient, C_p, for three different unseen flow conditions, at one section.

Table 7.2: ONERA M6 wing: Reference aerodynamic coefficients and values computed with the predicted meshes.

	$M_\infty = 0.660, \alpha = 1.91°$		$M_\infty = 0.885, \alpha = 2.13°$		$M_\infty = 0.809, \alpha = 7.12°$	
	Reference	Predicted	Reference	Predicted	Reference	Predicted
C_L	0.146	0.147	0.240	0.241	0.645	0.646
C_D	0.0014	0.0013	0.0142	0.0143	0.0628	0.0630

cases.

7.2.2 Anisotropic spacing predictions on a geometrically parametrised aircraft

The second example considers the anisotropic spacing prediction for a full aircraft configuration that is geometrically parametrised at fixed transonic flow conditions corresponding to free-stream Mach number $M_\infty = 0.78$ and angle of attack $\alpha = 2.0°$.

The wings of the aircraft are parametrised using 11 geometric parameters, with the details given in Table 7.3 and for a fixed total semi-span of 8.15m.

To illustrate the variability in the geometry and the corresponding solution induced by the selected geometric parameters, Figure 7.9 depicts the pressure coefficient for three different geometric configurations. In the three cases shown, a strong shock wave is observed on the upper surface of the wing, clearly displaying the transonic nature

Table 7.3: Geometrically parametrised aircraft: design parameters and the range of variation for each parameter.

Parameter	**Lower Limit**	**Upper Limit**
Inboard sweep	$15°$	$50°$
Outboard sweep	$15°$	$60°$
Inboard semi-span	2.0 m	5.0 m
Inboard dihedral	$-7.5°$	$7.5°$
Outboard dihedral	$-7.5°$	$7.5°$
Mid-span twist	$0°$	$3°$
Tip twist	$-2°$	$3°$
Mid-span chord ratio	0.25	0.75
Tip chord ratio	0.25	0.75
Mid-span thickness ratio	0.25	0.75
Tip thickness ratio	0.25	0.75

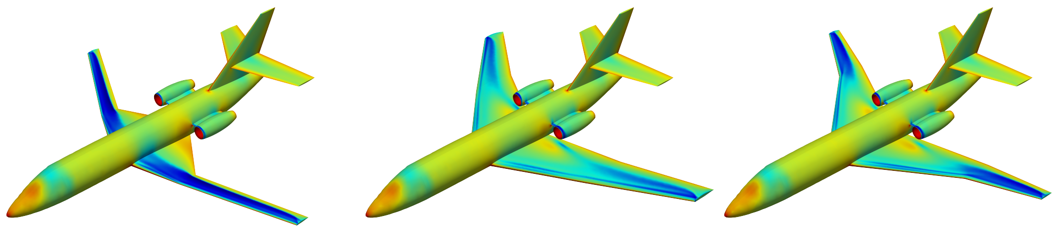
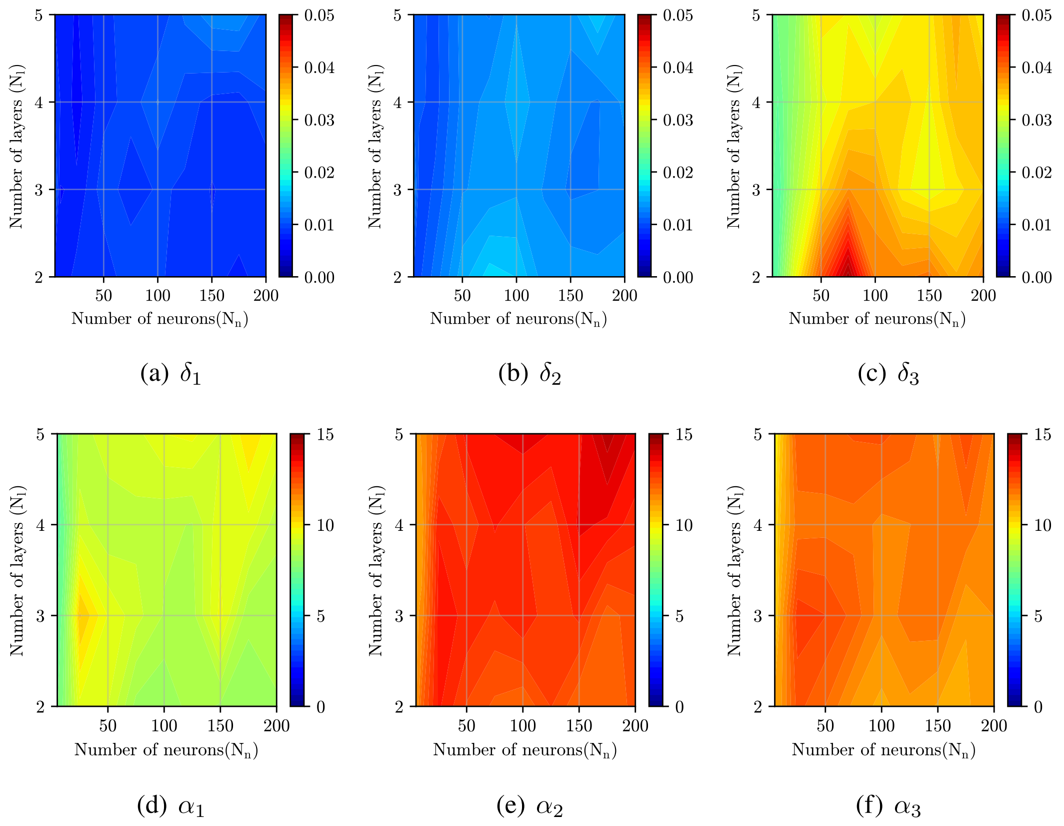

(a) Geometry 1 (b) Geometry 2 (c) Geometry 3

Figure 7.9: Geometrically parametrised aircraft: Pressure coefficient, C_p, for three different geometric configurations.

of the flow. With the first geometry, a well-defined shock wave is observed along the entire upper surface of the wing. With the second geometry, the shock structure changes significantly and the interaction of the wing and engine intake can be observed. Lastly, with the third geometry, a stronger shock is present over the outer section of the wing, and a complex interaction of the wing and engine is observed. These examples highlight the significant impact of the geometry on the flow features and emphasise the importance of using meshes tailored for each simulation.

As in the previous example, the strategy presented in the previous section is used to compute the metric tensor for each available simulation and to transfer the spacing to the background mesh, which in this example has approximately 847K elements and 143K nodes.

The NNs are trained using the same parameters as in the previous example, as described



(a) δ_1 (b) δ_2 (c) δ_3

(d) α_1 (e) α_2 (f) α_3

Figure 7.10: Geometrically parametrised aircraft: MAE for the spacings (δ_i) and angles (α_i) as a function of the number of layers and number of neurons in each layer employing the first model of Table 7.1.

in Section 7.1.4. Given the better performance and efficiency of the first model described in Table 7.1, this is the only model considered in this example, where two NNs are trained separately to predict spacing and anisotropy directions. Figure 7.10 shows the MAE for NNs with an increasing number of hidden layers and neurons, when using $N_{tr} = 40$. The results show a qualitative behaviour similar to the previous example despite the different nature of the parameters and the increased dimensionality of the problem. The prediction of the spacing is again significantly more accurate than the prediction of the angles that define the directions of anisotropy. The main difference with respect to the previous example is that the prediction of the angles that define the anisotropic direction is more challenging, in particular the second angle, α_2. This is expected given the fact that the same amount of data is utilised, $N_{tr} = 40$, but this problem has 11 geometric parameters.

To study the effect of the training dataset size, Figure 7.11 the maximum MAE for all

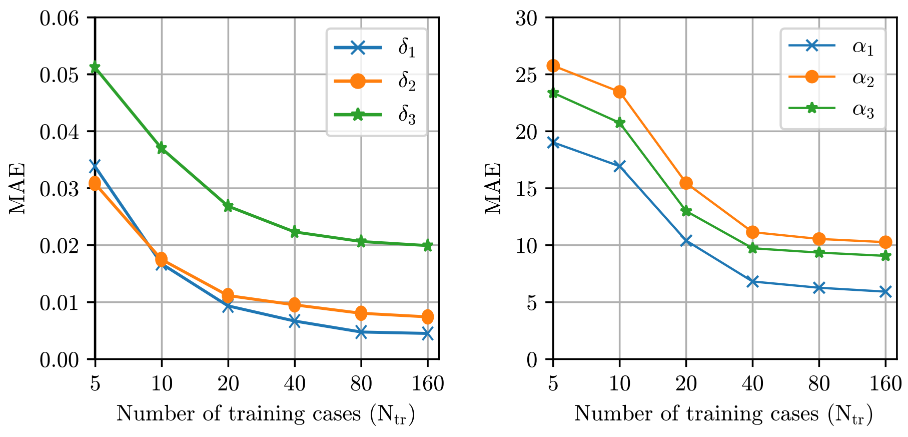
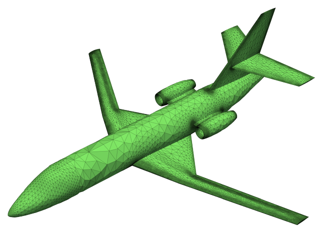


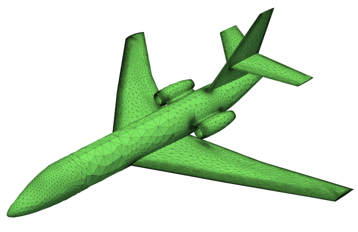
Figure 7.11: Geometrically parametrised aircraft: MAE for the spacings (δ_i) and angles (α_i) as a function of the number of training cases, N_{tr}, employing the first model of Table 7.1.

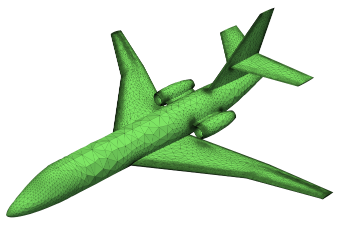
test cases as a function of the number of training cases for the six predicted outputs. As in the previous example the error decreases as the number of training cases increases. Despite the increased difficulty of this problem, it is remarkable to observe a significant gain in accuracy with only 40 training cases.

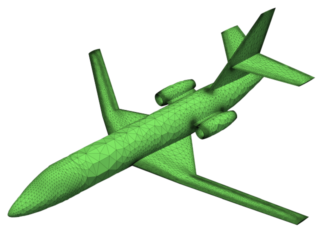
To illustrate the ability of the proposed approach to predict anisotropic spacing for a more complex problem, the trained NNs are used to predict the metric tensor at each point of the domain, which is subsequently used to generate an anisotropic mesh for three cases unseen during training. Figure 7.12 compares the target and predicted meshes for the three cases. The results clearly show the ability of the trained NN to accurately predict the regions where refinement is required as well as the anisotropic character of the target spacing.

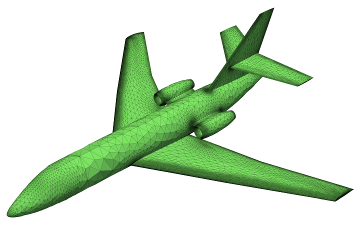
A more quantitative analysis is provided in Figure 7.13, showing the histogram of the anisotropic spacing accuracy for different training datasets. The histograms show that with only 20 training cases it is possible to predict 90% of the nodes with an acceptable spacing in the two first directions. It is remarkable that the accuracy in the first two directions is similar to the one obtained in the previous example, despite in this scenario 11 geometric parameters are considered and in the previous example only two flow conditions were used. The main difference between the two examples is that


(a) Geometry 1


(b) Geometry 2


(c) Geometry 3


(d) Geometry 1


(e) Geometry 2

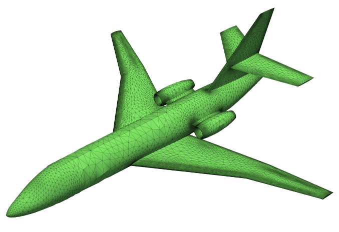
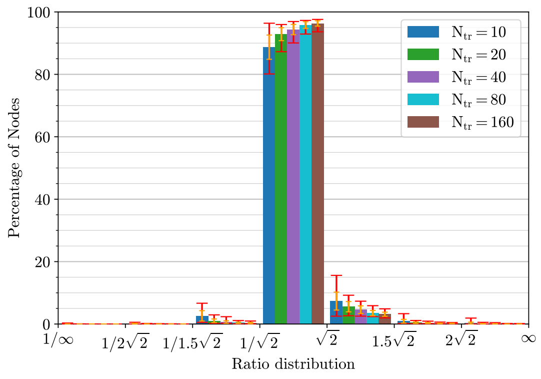
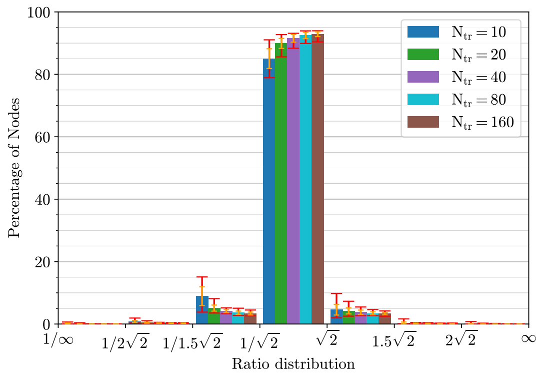

(f) Geometry 3

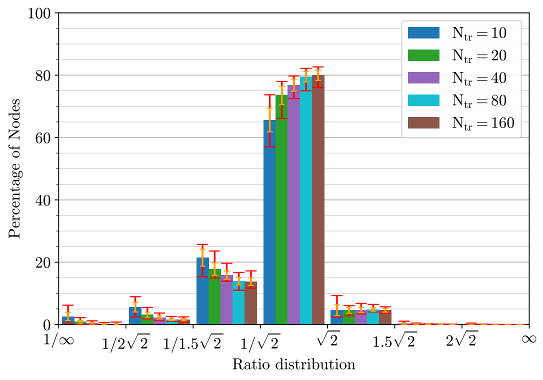
Figure 7.12: Geometrically parametrised aircraft: Target (top) and predicted (bottom) meshes for three geometric configurations unseen by the NN during training.



(a) Spacing in $\mathbf{e}_1$



(b) Spacing in $\mathbf{e}_2$



(c) Spacing in $\mathbf{e}_3$

Figure 7.13: Geometrically parametrised aircraft: Histogram of the ratio between the predicted and target spacings for an increasing number of training cases, N_{tr}.

in the current example it is particularly challenging to predict the spacing in the third direction. However, it is worth noting that this direction corresponds to the maximum of the three spacings and therefore the least relevant. Furthermore, the histograms show that the trained NN tends to refine more than required in this direction rather than under-refine, which is obviously preferable.

7.3 Concluding Remarks

This chapter has extended the background mesh-based learning framework to predict fully anisotropic spacing fields across a parametric design space involving both geometric and flow variations. By incorporating a morphing procedure to map background meshes from a reference geometry onto varied configurations, the framework achieves consistent data representation, enabling the prediction of spatially varying metric tensors through neural networks.

The methodology demonstrated strong performance in generating anisotropic meshes that adapt effectively to complex flow features such as shocks and expansions, even in configurations not included in the training set. The use of metric tensors allows directional refinement to be prescribed explicitly, resulting in higher mesh efficiency and improved alignment with flow structures. The predictions generalised well across a wide range of test cases, validating the robustness of the framework and its suitability for use in simulation-driven design settings.

A particular strength of the approach lies in its generality: the method accommodates arbitrary geometric changes without requiring topological remeshing and leverages a compact and structured representation of anisotropy. Additionally, the use of a shared background mesh simplifies the neural network architecture and data pipeline, making the approach scalable to high-dimensional parametric studies.

However, the approach is not without limitations. The accuracy of the morphing procedure, while generally sufficient, can degrade for extreme geometric deformations, potentially introducing localised distortion in the interpolated spacing fields. Furthermore, the fixed background mesh introduces a trade-off between spatial resolution and model complexity: finer meshes allow greater detail in the predicted spacing field but increase the number of outputs and training burden. These factors highlight the importance of careful mesh design and suggest avenues for future work, such as adaptive background meshes or hybrid representations that combine the strengths of both coarse and fine resolutions.

Chapter 8

Concluding Remarks

8.1 Summary of thesis achievements

This thesis has presented a series of data-driven strategies to enable the automatic generation of near-optimal unstructured meshes for CFD applications. The core objective was to leverage historical high-fidelity simulation data to train neural networks capable of predicting mesh spacing fields for new, unseen cases—thereby reducing reliance on manual refinement and improving both efficiency and consistency in the simulation pipeline.

Three complementary methodologies were developed, each progressively extending the complexity and generality of the mesh spacing prediction.

The first approach introduced a method for predicting the characteristics of mesh sources – the position, spacing, and radius of influence – used in unstructured mesh generation algorithms. A systematic framework was proposed to extract these sources from a database of prior simulations and train a neural network to predict their properties for new geometric or flow configurations. The resulting spacing fields were shown to yield near-optimal meshes capable of capturing complex flow features with high accuracy. In addition, a source-merging strategy was introduced to improve efficiency by reducing redundancy in the predicted source set. Numerical experiments demonstrated that the approach produced meshes that delivered aerodynamic quantities within accepted tolerances for industrial applications, even with limited training data.

Building on this, a second strategy was developed based on a background mesh representation. Rather than learning source characteristics, this method directly predicted the spacing at each node of a coarse background mesh. A conservative interpolation strategy was devised to map spacing data from existing solution meshes onto the background mesh, enabling consistent data generation for training. The trained neural network then predicted spacing fields for new cases, which could be passed to a standard meshing tool. Compared to the source-based method, this approach required less training data and simpler model architectures while maintaining prediction accuracy. Applications to three-dimensional flow problems confirmed its effectiveness and scalability.

The final contribution introduced two significant advancements. First, the method was extended to accommodate geometric variability across simulations. A mesh morphing and mapping strategy was developed to enable consistent representation of spacing fields across parametrised geometries. Additionally, the conservative nature of previous work was kept, through the use of metric intersection, ensuring the generated meshes would capture all the features. Second, the method was generalised to predict fully anisotropic spacing fields through the use of metric tensors. By training neural networks to learn both anisotropic directions and magnitudes, the approach allowed for precise, directionally sensitive mesh adaptation – particularly important in resolving features such as shocks. These capabilities were demonstrated using numerical examples – ranging from parametrised wings to full aircraft configurations with 11 geometric parameters, demonstrating that the method could predict critical anisotropic directions with high accuracy, even with limited training data, and produce meshes that delivered reliable simulation results.

Throughout all approaches, the environmental and computational benefits of AI-driven meshing were considered. While a detailed analysis of carbon footprint was carried out for the source-based method, the insights gained informed the design and evaluation of the subsequent strategies, reinforcing the potential of data-driven meshing to reduce computational cost and environmental impact across workflows.

Together, these contributions form a unified framework for data-driven mesh generation—one that embeds engineering knowledge from prior CFD analyses into predictive tools that enhance simulation efficiency, consistency, and sustainability.

8.2 Ethical Considerations in the Use of AI for Mesh Generation

The increasing adoption of AI technologies in engineering design processes raises legitimate ethical considerations, particularly regarding the potential displacement of skilled professionals by automation. In the context of mesh generation, a traditionally manual and expertise-driven task, concerns may arise that the automation of this process could diminish the role of human mesh designers or even replace them entirely.

This thesis acknowledges these concerns, but offers a more nuanced view grounded in the realities of industrial practice. Rather than eliminating roles, AI-driven mesh prediction serves to enhance productivity and shift the role of engineers towards higher-level decision-making and innovation. In workflows where the same engineer is responsible for both design and meshing, the automation of mesh generation reduces the time spent on routine, iterative refinement. This enables engineers to focus more on design exploration, sensitivity studies, and innovation—activities that directly contribute to improved product performance and competitiveness.

In larger industrial environments, where dedicated specialists are responsible for meshing tasks, the introduction of predictive tools does not eliminate their function but augments it. These tools offer rapid initial estimates that can serve as starting points for more detailed manual refinement where required, allowing experienced meshing engineers to allocate their time more effectively and handle a larger number of cases with improved turnaround. Furthermore, this shift reduces bottlenecks in simulation pipelines and enables more extensive design-space exploration under tight time constraints, thereby accelerating development cycles.

When used appropriately, these tools support more efficient simulation workflows and reduce the routine effort involved in mesh generation, enabling engineers to dedicate more time to design assessment, exploration, and decision-making—areas where human-led engineering judgement and experience remain essential.

8.3 Future Work

While this thesis has developed and demonstrated a robust framework for data-driven mesh generation, several directions remain for future exploration and improvement. These can be broadly categorised into methodological extensions and enhancements to the neural network component.

8.3.1 Methodology Extensions

As described in Section 6.3, the background mesh used to train the neural network is currently a simple mesh that was defined manually. While this approach has proven effective, a fully general and scalable method would prefer an automated strategy to generate background meshes tailored to the specific characteristics of a given training dataset. Such a method would need to balance coverage of relevant design parameters, geometric variability, and resolution requirements while remaining computationally tractable.

Another promising avenue is the integration of the two strategies explored in this thesis: point-source spacing and background mesh predictions. While both have their own advantages, a hybrid approach could exploit the flexibility of source-based representations and the spatial regularity of background meshes.

8.3.2 Neural Network Improvements

This thesis has focused primarily on developing and validating the core methodology, using feed-forward neural networks (FNNs) as a baseline. While effective, several opportunities exist to improve predictive accuracy, data efficiency, and scalability through more advanced machine-learning techniques.

One such avenue is transfer learning, which could enable trained models to adapt to new geometries or flow conditions with minimal additional data or retraining. This is particularly appealing in industrial settings, where frequent design variations and limited labelled data are common.

Alternative neural network architectures may also enhance performance. For example, autoencoders could be used to encode anisotropic spacing fields into compact latent representations, capturing directional structure in a lower-dimensional space. These latent vectors could then be predicted using a lightweight feed-forward network, simplifying the regression task and potentially improving the accuracy and robustness of the method.

Other architectures may offer advantages for specific aspects of the problem. Convolutional Neural Networks (CNNs) have been explored in the context of mesh prediction by projecting spacing fields onto structured grids in two dimensions; however, their applicability is limited in three-dimensional, unstructured domains. In contrast, Graph Neural Networks (GNNs) naturally operate on mesh-based data and could directly learn from nodal connectivity and spatial relationships, making them a promising candidate for future exploration.

While these architectures come with additional complexity and data requirements, they offer promising directions for improving prediction quality, especially for highly anisotropic or large-scale mesh problems. Exploring and comparing these models would be a valuable extension of the current work.

8.3.3 Further Opportunities for Development

While this thesis has focused on applications within the aerospace sector, particularly for compressible flow simulations, the underlying methodology is broadly applicable. Future work could explore its extension to other simulation domains such as solid mechanics, acoustics, electromagnetics, or thermal analysis—any context in which mesh quality directly impacts solution accuracy. Integrating such AI-driven mesh prediction into a wider range of simulation workflows would support more efficient design processes across engineering disciplines, particularly where rapid iteration and performance evaluation are critical.

Another promising direction is to integrate the predicted mesh spacing fields more closely with conventional mesh adaptation frameworks. For instance, these predictions could be used to initialise the mesh in a standard adaptation process, providing a strong starting point that may reduce the number of refinement cycles needed. This combined approach could improve the efficiency and robustness of meshing, particularly in cases with complex geometries or flow conditions.

A key extension of the current framework is to establish a more explicit link between mesh resolution and the desired accuracy of the simulation. Rather than predicting a generic high-resolution mesh, it would be desirable to control the mesh according to specified error tolerances. This would allow users to tailor the mesh density based on the stage of the design cycle—for example, using coarser meshes during early design exploration, and finer meshes when higher accuracy is required for final validation. Incorporating an error-to-mesh mapping would further increase the practicality of data-driven mesh generation in real-world workflows.

The current work has focused on steady-state, inviscid simulations. Extending the methodology to handle unsteady and viscous flows would represent a significant step forward. These problems introduce time-dependent or boundary-layer-driven features that often demand higher mesh resolution and more adaptive strategies.

Finally, there is strong industrial interest in developing optimal grouped meshes—meshes that remain effective across a range of flow conditions or design parameters. Rather than generating a new mesh for every possible scenario, the goal would be to construct a single mesh that maintains acceptable accuracy across a neighbourhood in the parameter space. Achieving this would require strategies for identifying representative cases and combining their spacing requirements into a single, conservative mesh. The potential for such meshes to reduce simulation overhead and accelerate early-stage design makes this a highly compelling direction for future work.

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). TensorFlow: a system for Large-Scale machine learning. In 12th USENIX symposium on operating systems design and implementation (OSDI 16), pages 265–283.

Aggarwal, C. C. (2023). Neural networks and deep learning. Springer International Publishing, Cham, Switzerland, 2 edition.

Alauzet, F. and Loseille, A. (2010). High-order sonic boom modeling based on adaptive methods. J. Comput. Phys., 229(3):561–593.

Alfonzetti, S., Coco, S., Cavalieri, S., and Malgeri, M. (1996). Automatic mesh generation by the let-it-grow neural network. IEEE transactions on magnetics, 32(3):1349–1352.

Anderson, J. D. (2016). Fundamentals of aerodynamics. McGraw-Hill Education, Columbus, OH, 6 edition.

Balla, K., Sevilla, R., Hassan, O., and Morgan, K. (2021). An application of neural networks to the prediction of aerodynamic coefficients of aerofoils and wings. Applied Mathematical Modelling, 96:456–479.

Bengio, Y. (2016). Deep Learning. Adaptive Computation and Machine Learning series. MIT Press, London, England.

Bishop, C. M. and Clements, P. (2006). Pattern Recognition and Machine Learning. Information Science and Statistics. Springer, New York, NY, 1 edition.

Bohn, J. and Feischl, M. (2021). Recurrent neural networks as optimal mesh refinement strategies. Computers & Mathematics with Applications, 97:61–76.

Cai, S., Mao, Z., Wang, Z., Yin, M., and Karniadakis, G. E. (2022). Physics-informed neural networks (PINNs) for fluid mechanics: A review. Acta Mechanica Sinica, pages 1–12.

Chedid, R. and Najjar, N. (1996). Automatic finite-element mesh generation using artificial neural networks-part i: Prediction of mesh density. IEEE Transactions on Magnetics, 32(5):5173–5178.

Chen, G. and Fidkowski, K. (2020). Output-based error estimation and mesh adaptation using convolutional neural networks: Application to a scalar advection-diffusion problem. In AIAA Scitech 2020 Forum, page 1143.

Chen, X., Liu, J., Gong, C., Li, S., Pang, Y., and Chen, B. (2021). MVE-Net: An automatic 3-D structured mesh validity evaluation framework using deep neural networks. Computer-Aided Design, 141:103104.

Cheng, J. and Druzdzel, M. J. (2000). Computational investigation of low-discrepancy sequences in simulation algorithms for bayesian networks.

Cheng, S.-W., Dey, T. K., and Shewchuk, J. (2009). Delaunay Mesh Generation. Chapman & Hall/CRC Computer & Information Science Series. Chapman & Hall/CRC, Philadelphia, PA.

Coons, S. (1976). Surfaces for Computer-aided Design of Space Forms. Project MAC, Massachusetts Institute of Technology.

Dawes, W., Dhanasekaran, P., Demargne, A., Kellar, W., and Savill, A. (2001). Reducing bottlenecks in the CAD-to-mesh-to-solution cycle time to allow CFD to participate in design. Journal of Turbomachinery, 123(3):552–557.

Douglas, S. C. and Yu, J. (2018). Why relu units sometimes die: Analysis of single-unit error backpropagation in neural networks. CoRR, abs/1812.05981.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(61):2121–2159.

Dyck, D., Lowther, D., and McFee, S. (1992). Determining an approximate finite element mesh density using neural network techniques. IEEE transactions on magnetics, 28(2):1767–1770.

Faux, I. D. and Pratt, M. J. (1980). Computational Geometry for Design and Manufacture. John Wiley & Sons, Nashville, TN.

Ferguson, J. (1964). Multivariable curve interpolation. J. ACM, 11:221–228.

Giacomini, M., Cortellessa, D., Vieira, L. M., Sevilla, R., and Huerta, A. (2025). A hybrid pressure formulation of the face-centred finite volume method for viscous laminar incompressible flows. Int. J. Numer. Methods Eng., 126(10).

Glantz, S. A. and Slinker, B. K. (2001). Primer of applied regression & analysis of variance, ed, volume 654. McGraw-Hill, Inc., New York.

Glantz, S. A., Slinker, B. K., and Neilands, T. B. (2015). Primer of applied regression & analysis of variance, third edition. McGraw-Hill Professional, New York, NY, 3 edition.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., editors, Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, volume 9 of Proceedings of Machine Learning Research, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

Golub, G. H., Hansen, P. C., and O'Leary, D. P. (1999). Tikhonov regularization and total least squares. SIAM journal on matrix analysis and applications, 21(1):185–194.

Hagan, M. T., Demuth, H. B., and Beale, M. (1997). Neural network design. PWS Publishing Co.

Halton, J. H. (1964). Algorithm 247: Radical-inverse quasi-random point sequence. Communications of the ACM, 7(12):701–702.

Hassan, O. (2025). Personal communication.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). The elements of statistical learning. Springer series in statistics. Springer, New York, NY, 2 edition.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. CoRR, abs/1502.01852.

Hirsch, C. (2007). Numerical computation of internal and external flows: The fundamentals of computational fluid dynamics. Butterworth-Heinemann, Oxford, England, 2 edition.

Ian Goodfellow, Y. B. and Courville, A. (2016). Deep Learning. Adaptive Computation and Machine Learning series. MIT Press, London, England.

Karman, S. L., Wyman, N., and Steinbrenner, J. P. (2017). Mesh generation challenges: A commercial software perspective. In 23rd AIAA Computational Fluid Dynamics Conference, page 3790.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

Lacoste, A., Luccioni, A., Schmidt, V., and Dandres, T. (2019). Quantifying the carbon emissions of machine learning. arXiv preprint arXiv:1910.09700.

Lannelongue, L., Grealey, J., and Inouye, M. (2021). Green algorithms: quantifying the carbon footprint of computation. Advanced science, 8(12):2100707.

Lawson, C. L. (1977). Software for C1 surface interpolation. In Mathematical Software, pages 161–194. Elsevier.

Lee, D. and Lin, A. (1986). Generalized delaunay triangulation for planar graphs. Discrete Computational Geometry, 1(1):201 – 217. Cited by: 211; All Open Access, Bronze Open Access.

Lock, C., Hassan, O., Sevilla, R., and Jones, J. (2023). Meshing using neural networks for improving the efficiency of computer modelling. Engineering with Computers, 39(6):3791–3820.

Lock, C., Hassan, O., Sevilla, R., and Jones, J. (2024). Predicting the near-optimal mesh spacing for a simulation using machine learning. In Ruiz-Gironés, E., Sevilla, R., and Moxey, D., editors, SIAM International Meshing Roundtable 2023, pages 115–136. Springer Nature Switzerland.

Loseille, A., Dervieux, A., and Alauzet, F. (2010). Fully anisotropic goal-oriented mesh adaptation for 3D steady Euler equations. Journal of Computational Physics, 229(8):2866–2897.

Manevitz, L., Bitar, A., and Givoli, D. (2005). Neural network time series forecasting of finite-element mesh adaptation. Neurocomputing, 63:447–463.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 5(4):115–133.

Michal, T. (2019). Development of an anisotropic solution adaptive meshing tool for production aerospace applications. In Sixth Workshop on Grid Generation for Numerical Computations.

Minsky, M. and Papert, S. A. (1987). Perceptrons : An Introduction to Computational Geometry,. Perceptrons. MIT Press, London, England.

Ni, R.-H. (1982). A multiple-grid scheme for solving the euler equations. AIAA J., 20(11):1565–1571.

Peraire, J., Vahdati, M., Morgan, K., and Zienkiewicz, O. C. (1987). Adaptive remeshing for compressible flow computations. J. Comput. Phys., 72(2):449–466.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. W. (2020). Learning mesh-based simulation with graph networks. In International Conference on Learning Representations.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. U.S.S.R. Comput. Math. Math. Phys., 4(5):1–17.

Rasmussen, C. E. and Williams, C. K. I. (2005). Gaussian processes for machine learning. Adaptive Computation and Machine Learning series. MIT Press, London, England.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychol. Rev., 65(6):386–408.

Ruder, S. (2016). An overview of gradient descent optimization algorithms.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323:533–536.

Sanchez-Gamero, S., Hassan, O., and Sevilla, R. (2024). A machine learning approach to predict near-optimal meshes for turbulent compressible flow simulations. International Journal of Computational Fluid Dynamics, 38(2-3):221–245.

Schmitt, V. (1979). Pressure distributions on the ONERA M6-wing at transonic Mach numbers, experimental data base for computer program assessment. AGARD AR-138.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. (2020). Green ai. Communications of the ACM, 63(12):54–63.

Sevilla, R., Zlotnik, S., and Huerta, A. (2020). Solution of geometrically parametrised problems within a CAD environment via model order reduction. Computer methods in applied mechanics and engineering, 358:112631.

Shewchuk, J. R. (2002). Delaunay refinement algorithms for triangular mesh generation. Comput. Geom., 22(1-3):21–74.

Slotnick, J. P., Khodadoust, A., Alonso, J., Darmofal, D., Gropp, W., Lurie, E., and Mavriplis, D. J. (2014). CFD vision 2030 study: a path to revolutionary computational aerosciences. Technical report.

Sørensen, K., Hassan, O., Morgan, K., and Weatherill, N. (2003a). A multigrid accelerated hybrid unstructured mesh method for 3D compressible turbulent flow. Computational Mechanics, 31(1-2):101–114.

Sørensen, K. A., Hassan, O., Morgan, K., and Weatherill, N. P. (2003b). A multigrid accelerated hybrid unstructured mesh method for 3D compressible turbulent flow. Comput. Mech., 31(1-2):101–114.

Sørensen, K. A., Hassan, O., Morgan, K., and Weatherill, N. P. (2003c). A multigrid accelerated time-accurate inviscid compressible fluid flow solution algorithm employing mesh movement and local remeshing. Int. J. Numer. Methods Fluids, 43(5):517–536.

Stüben, K. (2001). A review of algebraic multigrid. J. Comput. Appl. Math., 128(1-2):281–309.

Swanson, R. and Turkel, E. (1997). Multistage schemes with multigrid for euler and navier-stokes equations.

Tamura, Y. and Fujii, K. (1993). Conservation law for moving and transformed grids. In 11th Computational Fluid Dynamics Conference, Reston, Virigina. American Institute of Aeronautics and Astronautics.

Thompson, J. F., Soni, B. K., and Weatherill, N. P. (1998). Handbook of grid generation. CRC press.

Vandewoestyne, B. and Cools, R. (2006). Good permutations for deterministic scrambled Halton sequences in terms of L2-discrepancy. Journal of Computational and Applied Mathematics, 189(1):341–361.

Wesseling, P. and Oosterlee, C. W. (2001). Geometric multigrid with applications to computational fluid dynamics. J. Comput. Appl. Math., 128(1-2):311–334.

Yang, J., Dzanic, T., Petersen, B., Kudo, J., Mittal, K., Tomov, V., Camier, J.-S., Zhao, T., Zha, H., Kolev, T., et al. (2022). Reinforcement learning for adaptive mesh refinement. In International Conference on Learning Representations.

Zhang, Z., Jimack, P. K., and Wang, H. (2021). Meshingnet3D: Efficient generation of adapted tetrahedral meshes for computational mechanics. Advances in Engineering Software, 157:103021.

Zhang, Z. and Naga, A. (2005). A new finite element gradient recovery method: Superconvergence property. SIAM J. Sci. Comput., 26(4):1192–1213.

Zhang, Z., Wang, Y., Jimack, P. K., and Wang, H. (2020). Meshingnet: A new mesh generation method based on deep learning. In International Conference on Computational Science, pages 186–198. Springer.

Zienkiewicz, O. C. and Zhu, J. Z. (1992a). The superconvergent patch recovery and a posteriori error estimates. part 1: The recovery technique. International Journal for Numerical Methods in Engineering, 33(7):1331–1364.

Zienkiewicz, O. C. and Zhu, J. Z. (1992b). The superconvergent patch recovery and a posteriori error estimates. part 2: Error estimates and adaptivity. International Journal for Numerical Methods in Engineering, 33(7):1365–1382.

Appendix A

Error-Based Element Sizing

The computation of optimal element sizing is guided by error estimation techniques, which aim to determine the appropriate element sizes throughout the computational domain. The core objective is to place smaller elements in regions where numerical error is high, and larger elements where the error is low. Importantly, numerical error is often highly directional. For instance, across a shock – where sharp gradients occur, the error tends to be large, whereas tangential to the shock, the error is typically much smaller.

One approach is based on minimising interpolation error, as proposed by Loseille and Alauzet (Loseille et al., 2010). The method formulates mesh adaptation as a continuous optimisation problem, defining a Riemannian metric space in which mesh elements are characterised by a metric tensor, defined in Section 3.2.2. The adaptation seeks an optimal metric field that minimises interpolation error in the L_p norm, subject to a mesh complexity constraint. This leads to highly anisotropic meshes that efficiently capture flow features such as shocks and vortices. The optimal metric tensor is derived from the Hessian of a chosen flow variable, ensuring alignment with solution anisotropy.

A.1 Error Indicator

In this work, an alternative error-based to determining the ideal sizing approach is used to compute the ideal mesh spacing, the use of error indicator (Peraire et al., 1987; Thompson et al., 1998). In a one-dimensional setting, the exact solution, $\sigma_{\text{exact}}(x)$, can be approximated by a piecewise linear function, $\hat{\sigma}(x)$. The error at a given location x is

$$E(x) = \sigma(x) - \sigma_{\text{exact}}(x). \tag{A.1}$$

Since the exact solution is unknown, it is approximated using a second-order polynomial, $\sigma^*(x)$. This error vanishes for linear solutions, while smooth nonlinear solutions can be approximated with arbitrary precision using higher-order polynomials. Rather than employing computationally expensive quadratic finite elements, an alternative approach reconstructs a quadratic approximation from the linear solution. Assuming that the nodal values of both the quadratic and linear solutions coincide, the error within an element e is expressed as

$$E_e(\zeta) = \frac{1}{2}\zeta(h_e - \zeta)\left.\frac{d^2\sigma^*}{dx^2}\right|_e, \tag{A.2}$$

where ζ is the local element coordinate, h_e is the element length, and the second derivative is that of the second-order polynomial.

To quantify the error across the element, the root-mean-square (RMS) error is computed as

$$E_e^{\text{RMS}} = \left[\int_0^{h_e} \frac{E_e^2}{h_e} d\zeta\right]^{\frac{1}{2}} = \frac{1}{\sqrt{120}} h_e^2 \left|\frac{d^2\sigma^*}{dx^2}\right|_e. \tag{A.3}$$

The optimal mesh seeks to equidistribute the RMS error across all elements, ensuring uniform error throughout the domain. This condition leads to the requirement

$$h_e^2 \left|\frac{d^2\sigma}{dx^2}\right| = K, \tag{A.4}$$

where K is a positive constant. The optimal element spacing is then given by

$$\delta = \sqrt{\frac{K}{\left|\frac{d^2\sigma}{dx^2}\right|}}. \tag{A.5}$$

This one-dimensional principle extends naturally to three dimensions, where the optimal spacing δ_β in a given direction β is

$$\delta_{i,\beta}^2 \left(\sum_{k,l=1}^{3} (H_i)_{kl} \beta^k \beta^l \right) = K, \tag{A.6}$$

where $(H_i)_{kl}$ represents the components of the Hessian matrix at node i,

$$(H_i)_{kl} = \frac{\partial^2 \sigma_i}{\partial x_k \partial x_l}. \tag{A.7}$$

The computation of mesh spacing relies on knowing the Hessian of a selected flow variable to determine the optimal element sizes and orientations. However, obtaining an accurate Hessian from numerical solutions presents challenges due to the discrete nature of the solution field. Using finite volume method with linear elements, means the computed solution is piecewise linear, leading to discontinuous first derivatives and undefined second derivatives within each element. Meaning, the Hessian needs to be recovered.

A.2 Computation of the Hessian matrix

To compute the second derivatives of a solution on an unstructured mesh, several recovery methods have been proposed. One approach is to use a vertex-centred finite volume approximation—i.e., employing a dual mesh—to estimate derivatives at the nodes (Sørensen et al., 2003a). Alternatively, a least-squares polynomial fitting can be applied to approximate derivatives at a given node (Zhang and Naga, 2005). Another method uses Gauss's theorem to integrate gradient fluxes across element boundaries, yielding second derivative approximations (Alauzet and Loseille, 2010).

In this work, a standard recovery technique originally developed for the finite element method is adopted (Zienkiewicz and Zhu, 1992a,b), employing a weighted averaging strategy to reconstruct the Hessian matrix. Although the flow field is solved using a vertex-centred finite volume method, it is noted that on tetrahedral meshes this approach is equivalent to the finite element method.

The process begins by estimating the gradient at each node using a weighted average of gradients from the surrounding elements. For a given node i, the recovered gradient $\boldsymbol{\nabla}\sigma_i$ is computed as

$$\boldsymbol{\nabla}\sigma_i \approx \frac{\sum\limits_{\Omega_e \in \mathcal{P}_i} |\Omega_e| \boldsymbol{\nabla}\sigma^{(e)}}{\sum\limits_{\Omega_e \in \mathcal{P}_i} |\Omega_e|} \tag{A.8}$$

where $\mathcal{P}_i$ denotes the patch of elements connected to node i, $\boldsymbol{\nabla}\sigma^{(e)}$ is the gradient within element Ω_e (computed from linear shape functions), and $|\Omega_e|$ is the volume of the element. This yields a continuous approximation of the first derivative at the node.

To recover the Hessian matrix, the same weighted averaging process is applied again, this time to the components of the recovered gradient, treating each as a scalar field. The resulting second derivatives form a continuous, piecewise-linear approximation of the Hessian matrix across the domain.

Appendix B

Total least-squares approximations in three dimensions

Finding the plane or line that produces the *best fit* to a given set of points is a classical problem in many applications of science and engineering. To define what is considered the best fit, it is necessary to specify a measure for the deviation of a point with respect to a plane or line. On many occasions, a simple measure that involves one of the coordinates is considered, but the result is highly dependent on the particular arrangement of the given set of points. This section describes the procedure to fit a plane or a line to a set of points in three dimensions by employing the true distance, measured in the orthogonal direction. This is a particular case of the technique usually referred to as the total least-squares.(Golub et al., 1999)

B.1 Fitting a plane to a set of points using orthogonal regression

Let us consider a set of n points in $\mathbb{R}^3$, denoted by $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1,\dots,n}$, and a plane, $\mathcal{P}$, defined by a point $\boldsymbol{x}_0 \in \mathbb{R}^3$ and a unit normal vector $\boldsymbol{n}$. By definition, a point $\boldsymbol{x} \in \mathcal{P}$

satisfies $\boldsymbol{n} \cdot (\boldsymbol{x} - \boldsymbol{x}_0) = 0$. A generic point $\boldsymbol{x}_i \in \mathcal{X}$ can be expressed as

$$\boldsymbol{x}_i = \boldsymbol{x}_0 + \lambda_i \boldsymbol{n} + \zeta_i \boldsymbol{t}, \tag{B.1}$$

where $\lambda_i = \boldsymbol{n} \cdot (\boldsymbol{x}_i - \boldsymbol{x}_0)$ is the distance from the point $\boldsymbol{x}_i$ to the plane $\mathcal{P}$, ζ_i is the distance from the orthogonal projection of $\boldsymbol{x}_i$ onto $\mathcal{P}$, namely $\hat{\boldsymbol{x}}_i$, to the point $\boldsymbol{x}_0$ and $\boldsymbol{t} = (\hat{\boldsymbol{x}}_i - \boldsymbol{x}_0)/\|\hat{\boldsymbol{x}}_i - \boldsymbol{x}_0)\|$.

To find the plane that best fits the given set of points, the point $\boldsymbol{x}_0$ and the unit normal vector $\boldsymbol{n}$ that minimises the energy function given by

$$E(\boldsymbol{x}_0, \boldsymbol{n}) = \sum_{i=1}^{n} \lambda_i^2 = \boldsymbol{n}^T \boldsymbol{M} \boldsymbol{n} \tag{B.2}$$

are sought, where $\boldsymbol{M} := \sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{x}_0)(\boldsymbol{x}_i - \boldsymbol{x}_0)^T$.

To simplify the minimisation problem, the point $\boldsymbol{x}_0$ is taken as the average position of the given set of points $\mathcal{X}$, namely $\boldsymbol{x}_0 = \sum_{i=1}^{n} \boldsymbol{x}_i/n$. With this choice, the energy function becomes a quadratic form and the unit normal vector $\boldsymbol{n}$ that provides the minimum of $E(\boldsymbol{n})$ is simply the normalised eigenvector corresponding to the minimum eigenvalue of the matrix $\boldsymbol{M}$.

B.2 Fitting a line to a set of points using orthogonal regression

Let us consider a set of n points in $\mathbb{R}^3$, denoted by $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1,\dots,n}$, and a line, ℓ, defined by a point $\boldsymbol{x}_0 \in \mathbb{R}^3$ and a unit vector $\boldsymbol{t}$. By definition, a point $\boldsymbol{x} \in \ell$ satisfies $\boldsymbol{t} \times (\boldsymbol{x} - \boldsymbol{x}_0) = \boldsymbol{0}$. A generic point $\boldsymbol{x}_i \in \mathcal{X}$ can be expressed as

$$\boldsymbol{x}_i = \boldsymbol{x}_0 + \lambda_i \boldsymbol{n} + \zeta_i \boldsymbol{t}, \tag{B.3}$$

where $\lambda_i = \boldsymbol{t} \cdot (\boldsymbol{x}_i - \boldsymbol{x}_0)$ is the distance from $\boldsymbol{x}_0$ to the orthogonal projection of $\boldsymbol{x}_i$ onto the line ℓ, namely $\hat{\boldsymbol{x}}_i$. The coefficient $\zeta_i = \|\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i\| = \|\boldsymbol{x}_i - \boldsymbol{x}_0 - \lambda_i \boldsymbol{t}\|$ represents the distance from $\boldsymbol{x}_i$ to its orthogonal projection onto the line ℓ.

To find the line that best fits the given set of points, the point $\boldsymbol{x}_0$ and the unit vector $\boldsymbol{t}$ that minimises the energy function given by

$$E(\boldsymbol{x}_0, \boldsymbol{t}) = \sum_{i=1}^{n} \zeta_i^2 = \boldsymbol{t}^T \widehat{\boldsymbol{M}} \boldsymbol{t} \tag{B.4}$$

are sought, where $\widehat{\boldsymbol{M}} = \mathrm{tr}(\boldsymbol{M})\boldsymbol{I} - \boldsymbol{M}$ and $\boldsymbol{M} := \sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{x}_0)(\boldsymbol{x}_i - \boldsymbol{x}_0)^T$.

To simplify the minimisation problem, the point $\boldsymbol{x}_0$ is taken as the average position of the given set of points $\mathcal{X}$, namely $\boldsymbol{x}_0 = \sum_{i=1}^{n} \boldsymbol{x}_i / n$. With this choice, the energy function becomes a quadratic form and the unit vector $\boldsymbol{t}$ that provides the minimum of $E(\boldsymbol{t})$ is the normalised eigenvector corresponding to the minimum eigenvalue of the matrix $\widehat{\boldsymbol{M}}$.

Appendix C

Neural Network Hyperparameter Tuning

This appendix presents a series of exploratory studies aimed at understanding the influence of key neural network (NN) hyperparameters on the training process used to predict mesh spacing. The goal of this investigation is to identify configurations that lead to improved convergence behaviour, reduced prediction error, and more stable training dynamics.

The hyperparameters examined include activation functions (as described in Section 4.2.2), and the optimisation algorithms (outlined in Section 4.3). Each plays a crucial role in the ability of the network to generalise and learn efficiently. To isolate the effect of each parameter, targeted comparisons are performed while keeping the remaining network settings fixed. Performance is assessed primarily via the R^2 score on unseen test cases.

The sections that follow summarise the results of these studies, offering insights into how specific hyperparameters impact network performance and guiding effective configuration strategies for the mesh spacing prediction problem.

C.1 Optimiser Study

The choice of optimisation algorithm can significantly affect both convergence behaviour and final model performance. This section examines the sensitivity of various optimisers in the context of predicting the five source characteristics. As this was the first hyperparameter tested, a stable and commonly used activation function (ReLU) was selected to isolate the effect of optimiser choice. The batch size was fixed at 8, balancing convergence stability with computational cost. The architecture used was Model 3 from Table 5.1, where each source characteristic was predicted independently, resulting in a simple single-output regression task.

Three stochastic gradient descent (SGD) configurations were tested, each using a different learning rate: 0.01, 0.001, and 0.004. In addition, the Adam optimiser was assessed using its default parameters. The learning rate value of 0.004 was selected after performing a sweep across a range of candidate values to identify the best-performing configuration. For each optimiser, a grid of architectures was tested by varying the number of layers N_l and the number of neurons per layer N_n, repeating each configuration five times to account for variability. As in previous studies, the goal was to identify the best-performing model in terms of the worst R^2 value achieved on the unseen test case — following the methodology in Section 5.2.1.

Three stochastic gradient descent (SGD) variants were tested, each with a different learning rate: 0.01, 0.001, and 0.004. The value of 0.004 was selected following a learning rate sweep to determine the best-performing configuration. In addition, the Adam optimiser was evaluated with its default settings. For each optimiser, a grid of architectures was sampled by varying the number of hidden layers (N_l) and neurons per layer (N_n), with five repetitions per configuration to account for variability. To assess the performance of the architectures, the goal was to identify the best-performing model in terms of the worst R^2 value achieved on the unseen test cases – following the methodology laid out in Section 5.2.1.

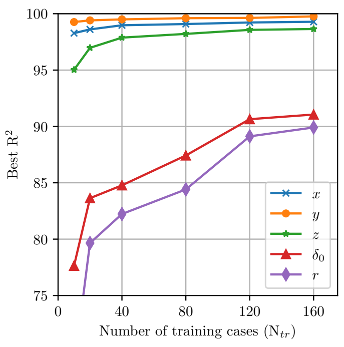
The results of this study are presented in Figure C.1. The performance of SGD is shown to be highly sensitive to the choice of learning rate, particularly when the number of training samples is small. At lower values of N_{tr}, inappropriate learning rates can significantly hinder model performance. However, as the size of the training set increases, this sensitivity is reduced, and performance tends to converge across different learning rates. When well-tuned, SGD can deliver the highest performance, as seen in the results for SGD(0.004). Nevertheless, this benefit comes at the cost of requiring careful hyperparameter tuning, which is computationally expensive – a key consideration in this work, and may not generalise well to different training scenarios.

In contrast, the Adam optimiser performs robustly across all training sizes and requires no manual learning rate tuning. While it may not always reach the absolute best result achieved by a tuned SGD configuration, Adam consistently delivers near-optimal performance, making it a compelling default choice. Its adaptability and low sensitivity to initialisation and learning rate make it well-suited for use across a range of architectures and problem settings, as discussed in Section 4.3.

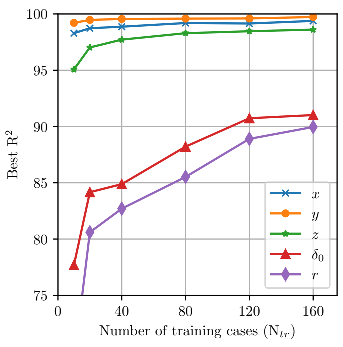
C.2 Activation Function Analysis

Following the optimiser study, this section investigates the effect of activation function choice. Model 3 from Table 5.1 is used again, where each source characteristic is predicted individually. Adam is adopted as the optimiser due to its strong performance and lack of tuning requirements, with a fixed batch size of 8. Again, the evaluation process and performance selection are detailed in Section 5.2.1.

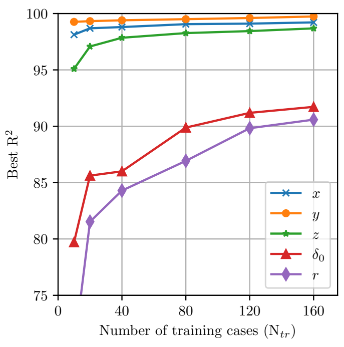
Figures C.2 and C.3 show hyperparameter performance maps for the activation functions ReLU, Sigmoid, and Tanh, with varying numbers of hidden layers (N_l) and neurons per layer (N_n). These results are based on the R^2 performance for the source radius r, which is typically the most challenging characteristic to predict accurately. The colour plots represent the lowest R^2 score obtained across the unseen test cases.



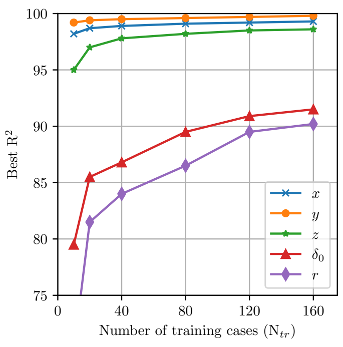
(a) SGD(0.01)



(b) SGD(0.001)



(c) SGD(0.004)



(d) Adam

Figure C.1: Best R^2 values obtained for each of the five source characteristics as a function of training set size N_{tr}, using the ReLU activation function and a batch size of 8. SGD results are shown for three different learning rates: (a) 0.01, (b) 0.001, and (c) 0.004. Subfigure (d) shows the performance using the Adam optimiser with default parameters. SGD can achieve strong performance when tuned, but Adam offers consistent near-optimal results without requiring learning rate tuning.

At $N_{tr} = 80$ (Figure C.2), the ReLU activation exhibits relatively consistent performance across all architectures, with little variation in R^2. This suggests that ReLU is largely insensitive to network depth and width. However, both Sigmoid and Tanh show a drop in performance for deeper networks, consistent with vanishing gradient issues. Nevertheless, these functions also demonstrate that, provided a sufficient range of architectures is sampled, they can outperform ReLU, achieving higher peak accuracy.

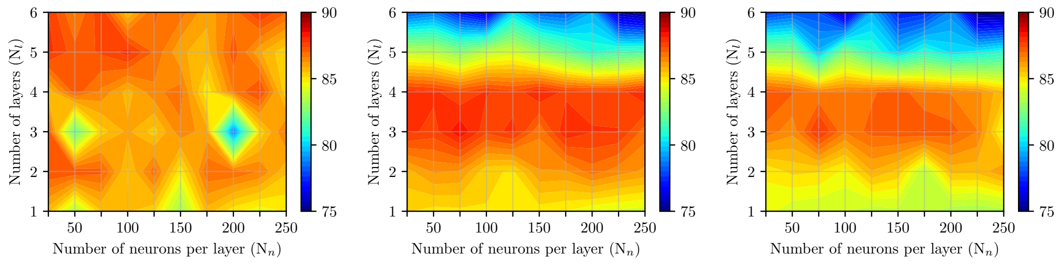
When the number of training cases is increased to $N_{tr} = 160$ (Figure C.3), the sensitivity of Sigmoid and Tanh to depth is reduced. The performance becomes more stable across the parameter space, suggesting that with more training data, deeper networks become more feasible. This supports the notion that vanishing gradients can be partially mitigated by increased data availability. Among the three activation functions, Sigmoid consistently yields the best performance when appropriate architectures are selected.

These findings are further supported in Figure C.4, which shows the best R^2 scores obtained for each source characteristic as a function of N_{tr}. The Sigmoid activation performs best across the most difficult outputs (δ_0, r), and all functions approach similar accuracy for the simpler spatial coordinates (x, y, z). The Tanh activation also performs well, while ReLU lags slightly behind, especially for more complex outputs.

Overall, this study highlights the importance of activation function choice and demonstrates that, despite their susceptibility to vanishing gradients in deep networks, smooth nonlinear activations like Sigmoid and Tanh can outperform ReLU, particularly when sufficient data and architectural diversity are available.

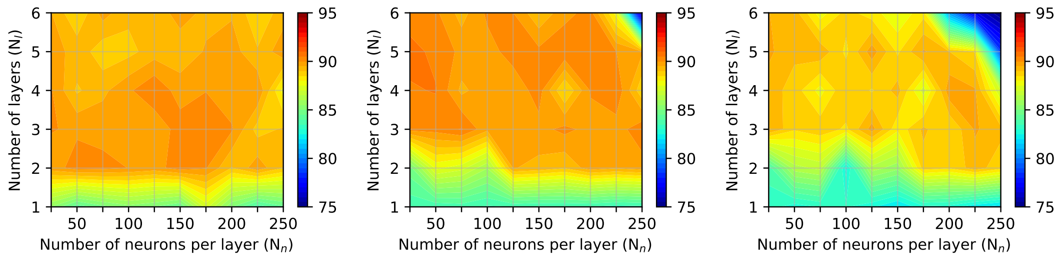
C.3 Conclusion

Based on these findings, the combination of the Adam optimiser with the Sigmoid activation function is identified as the most effective framework for predicting mesh spacing. This setup delivers consistently high performance without requiring extensive tuning, particularly when supported by sufficient architectural sampling. As such, this



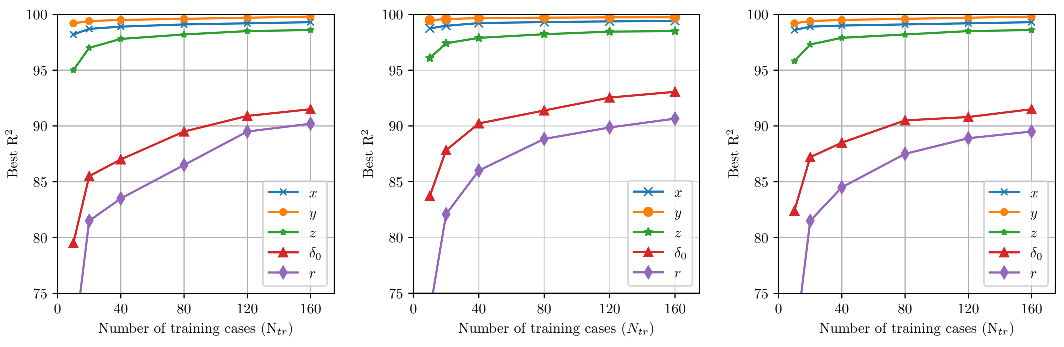
(a) ReLU (b) Sigmoid (c) Tanh

Figure C.2: Hyperparameter performance maps for different activation functions using $N_{tr} = 80$ training cases. The plots show the worst-case R^2 value across the unseen test cases for the source characteristic r, as a function of number of layers (N_l) and neurons per layer (N_n). ReLU maintains a relatively flat performance, whereas Sigmoid and Tanh show greater sensitivity to depth but achieve higher peak accuracy when properly configured.



(a) ReLU (b) Sigmoid (c) Tanh

Figure C.3: Hyperparameter performance maps for different activation functions using $N_{tr} = 160$ training cases. The increased data improves the stability of deeper architectures, particularly for Sigmoid and Tanh, which show reduced sensitivity to network depth. This enables higher consistency and accuracy across the tested network configurations.



(a) ReLU (b) Sigmoid (c) Tanh

Figure C.4: Best R^2 score achieved for each of the five predicted source characteristics as a function of the number of training cases N_{tr}. Sigmoid consistently outperforms ReLU and Tanh, especially for the more challenging outputs (δ_0 and r), while all activation functions achieve high accuracy on the simpler spatial components (x, y, z).

general configuration will be adopted throughout the remainder of this thesis for all neural network training tasks.