Computational Science and Engineering
(International Master's Program)

Technische Universität München

Master's Thesis

Exploratory Analysis of Turbulent Flow Data using GNN-based Surrogate Model

Vaishali Ravishankar

Computational Science and Engineering (International Master's Program)

Technische Universität München

Master's Thesis

Exploratory Analysis of Turbulent Flow Data using GNN-based Surrogate Model

Author:	Vaishali Ravishankar
1st examiner:	Univ.-Prof. Dr. Hans Joachim Bungartz
2nd examiner:	Prof. Dr. Jochen Garcke
Assistant advisor:	Kislaya Ravi, Christian Gscheidle
Submission Date:	April 15th, 2024

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

April 15th, 2024 Vaishali Ravishankar

Acknowledgments

I offer my deepest gratitude to my supervisors Christian Gscheidle and Arno Feiden at the Fraunhofer-Gesellschaft Scientific Computing and Algorithms Institute (SCAI). Since introducing and entrusting me with this intriguing research topic, your belief in my potential was a beacon that lit my way. No query, however trivial, went unanswered, as you stood by me every step of the way. Your availability and patience have been a source of immense encouragement, making this journey smoother and more meaningful. Your commitment to our weekly Tuesday check-ins and your deeply involved participation in our discussions consistently motivated me to push myself further and reach new heights.

I am profoundly grateful to Prof. Dr. Jochen Garcke for providing me with the opportunity to pursue my research at Fraunhofer SCAI and for being a pivotal figure during in this journey. His critical insights came at a crucial juncture of my research and his feedback was transformative, providing clarity and direction when most needed.

The resources and infrastructure provided by the SCAI institute, including access to the High-Performance Computing cluster and the suite of software, have been indispensable in the successful completion of my thesis, and I am deeply appreciative of it.

I extend my gratitude to Kislaya Ravi, whose role as a supervisor at the university extended far beyond traditional oversight. His willingness to lend an ear and offer timely advice proved invaluable in this endeavor.

I wish to express my appreciation to my friends, whose belief in me remained steadfast during moments of doubt and despair. Their encouragement and companionship offered solace through all the phases of this journey and kept my spirits high.

To my parents, Amma and Appa, I owe an immeasurable debt of gratitude for your unconditional love and unwavering support, which have been the cornerstone of my perseverance and success. Your constant encouragement and belief in me have been my greatest source of motivation. Thank you for everything.

Abstract

Turbulent flows, characterized by their complex and chaotic nature, play a pivotal role in various engineering and natural systems. Understanding and analyzing these phenomena is essential for optimizing design, predicting crucial outcomes and addressing real-world challenges. Therefore, obtaining accurate, efficient and rapid predictions of turbulent behaviors is of utmost importance. Data-driven methods such as deep learning algorithms are being increasingly implemented to speed up flow predictions compared to numerical solvers. However, these models tend to have poor generalization capabilities and are often restricted to simple geometries on structured grids. Hence, a Graph Neural Network (GNN) based surrogate model is proposed to handle unstructured mesh data of turbulent flow simulations. The underlying goal of this research is to leverage the predictions of the surrogate model to perform an exploratory analysis on the behavior of a High-speed Orienting Momentum with Enhanced Reversibility (HOMER) nozzle operating in turbulent flow conditions. Additionally, clustering and dimensional reduction techniques are employed to classify the various cases and phenomena occurring in this application, enhancing our understanding of turbulent nozzle flow dynamics.

Contents

Part I.

Introduction

1. Introduction

Fluid mechanics, a foundational branch of physics and engineering, underpins technological advancements in numerous critical sectors, including aerospace, automotive engineering, energy systems, environmental science, and biomedical engineering. The predictive modelling of fluid flow is central to the design, optimization, and operational efficiency of countless applications that fuel technological and scientific progress. At the heart of fluid mechanics are the Navier-Stokes Equations (NSE), a set of Partial Differential Equations (PDE) that describe the motion of fluids in space and time. Despite their foundational importance, these equations pose significant challenges due to their complexity, often defying exact analytical solutions and remaining one of the most perplexing open problems in mathematics [9]. These difficulties ushered in the era of Computational Fluid Dynamics (CFD), a transformative approach employing numerical methods to approximate solutions to the NSE. CFD has become an indispensable tool, leveraging advanced computing algorithms to simulate fluid flow scenarios. CFD still grapples with computational intensity and a suite of issues ranging from grid dependency to convergence challenges [10], underscoring a persistent need for innovation.

Turbulence modelling, in particular, stands out as a formidable challenge. Turbulence is characterized by velocity and pressure fluctuations across a diverse range of scales, from large vortices to minute eddies, compounded by numerical instabilities, especially in regions close to walls. While Direct Numerical Simulation (DNS) offers an avenue for precise modelling, its demand for extensive computational resources often renders it impractical for routine applications [28]. Consequently, in many practical scenarios, simplified turbulence models are employed, even though this comes at the expense of accuracy.

Advancements in the dynamically evolving field of CFD have seen substantial efforts channeled into enhancing turbulence models [34], improving meshing techniques [42], developing efficient Reduced Order Modelling (ROM) surrogates [1], and reducing computational complexity [40]. The pursuit of efficiency and optimization in CFD, marked by a growing interest in Machine Learning (ML) and Deep Learning (DL), has led to the application of these technologies for fluid dynamics. Unlike traditional computational methods, which are tethered to the constraints of processing power and simulation time, DL algorithms, once trained, can offer rapid predictions for new datasets. The integration of DL into CFD represents a promising frontier, with the potential to generate novel insights into fluid dynamics ([22], [36]).

Despite the advancements, DL methods have not yet been widely adopted in engineering practice, possibly due to the scarcity of extensive datasets and the poor generalization performance on previously unseen data [4]. This thesis leverages Graph Neural Networks

(GNN) to enable precise and efficient predictive models, specifically tailored for the nuanced application of nozzle flow simulation on unstructured meshes.

1.1. Literature review

In this section, we review significant contributions and advancements in the intersection of ML and CFD, with a particular focus on turbulence modelling. The foundational work in turbulence modelling can be traced back to Osborne Reynolds, who, in the late 19th century, pioneered the concept of turbulence through studies on fluid flow in pipes [37]. However, it was only in the 20th century that significant strides were made in formalizing turbulence models. The 1970s saw the introduction of the k-ϵ model [23], which improved RANS models by balancing accuracy with computational demands. Increased computational capabilities later enabled the adoption of DNS and Large Eddy Simulation (LES) [41] for more precise flow simulations. Recent trends show hybrid RANS-LES models emerging as a promising approach for accurate simulations.
ML algorithms have been increasingly deployed to construct surrogate models for complex turbulence systems. These surrogate models, embodying reduced-order representations, offer a streamlined computational alternative to exhaustive simulations. By the early 2010s, the exploration of ML in CFD began with significant contributions like Brunton's review [5], categorizing ML techniques into supervised, semi-supervised, and unsupervised learning. Supervised learning involves training models on well-labeled datasets to predict predefined outcomes, with regression algorithms playing a pivotal role in predicting continuous variables such as turbulence quantities. Semi-supervised learning, blending elements of supervised and unsupervised learning, has shown effectiveness in analyzing time-series data and images. Unsupervised learning focuses on unlabeled data and identifies patterns, clusters, or structures using methodologies like ROM and POD [2].
Beyond traditional classification, ML techniques can be discerned by their optimization goals into physics-informed and data-driven methods. Physics-informed methods integrate domain-specific knowledge, typically based on physical laws and principles, into the model formulation. They rely less on large amounts of data, as they primarily leverage the governing equations of the system. Termed Physics-Informed Neural Networks (PINN) [36], these methods are suitable when the physics of the problem is well understood and when data may be sparse or expensive to obtain.
In contrast, data-driven models aim to reduce the prediction error, relying on available data to make predictions or decisions. DL has facilitated the generation of data-driven closure terms for RANS and LES models, significantly elevating the precision of turbulence simulations [25]. Among the spectrum of DL methodologies, Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) have emerged as particularly potent in modelling turbulence. They adeptly capture and learn from the complex patterns inherent in turbulent flows. This capability extends to challenging scenarios such as non-equilibrium and multiphase flows. Milano and Koumoutsakos [27] explored the

use of neural networks for approximating flow fields around complex geometries, setting a precedent for DL in fluid mechanics. Tracey, Duraisamy, and Alonso ([44]) introduced machine learning techniques to modify turbulence models, illustrating ML's potential to refine simulation models. Zhang and Duraisamy's work [48] on data-driven turbulence closure models via multiscale Gaussian process regression paved the way for enhanced RANS equations.
Subsequent research efforts have further broadened the application of ML in CFD. More recently, Parish and Duraisamy [32] leveraged ML to inform turbulence model discrepancies, enhancing predictive capabilities. Ling, Kurzawski, and Templeton [25] significantly advanced RANS models' accuracy by incorporating Galilean invariance through deep neural networks. This period also witnessed the utilization of fully-convolutional neural networks by Guastoni et al. [13] for predicting velocity fields in turbulent flows, demonstrating the applicability of DL in fluid dynamics.
Expanding the scope to unstructured mesh data, recent advancements have explored graph-based and mesh-free techniques for fluid data representation. Trask et al. [46] introduced GMLS-Nets for mesh-free data analysis, demonstrating the versatility of ML approaches in handling complex data structures. Furthermore, Ogoke et al. [31] demonstrated the effectiveness of Graph Convolutional Neural Networks (GCNN) in predicting drag forces around airfoils. Liu et al. [26] showcased the effectiveness of GCNNs and meta-learning in predicting flow dynamics and enhancing turbulence models, highlighting the adaptive capabilities of GNNs in fluid dynamics.
In light of these developments, this research proposes a GNN-based surrogate model to analyze unstructured mesh data arising from turbulent flow within a High-speed Orienting Momentum with Enhanced Reversibility (HOMER) nozzle, developed by Michele Trancossi [45].

1.2. Scope and objectives

This thesis focuses on the cutting-edge intersection of DL, specifically GNNs, and CFD to enhance the predictive modelling of turbulent nozzle flow simulations. Traditionally, reaching stable solutions entails running simulations for extensive time intervals, a process that requires considerable computational resources and time. This thesis introduces surrogate models that circumvent the need for prolonged simulation times. By capturing transitional solutions at earlier stages, which take significantly less time, and processing them through a surrogate model, we can predict stable, steady-state solutions more efficiently. This approach utilizes GNNs as the core technology for the surrogate models, aiming to significantly reduce the computational burden associated with traditional CFD simulations. The research encapsulated within this thesis covers the development, evaluation, and practical application of a GNN-based surrogate model tailored for nozzle flow simulations. Furthermore, we employ data analysis techniques to categorize simulations based on velocity ratios and Coanda effect occurrences. The objectives set forth outline a

clear and structured path towards achieving the goals of this research and its outcomes.

1. **Develop a GNN model for predicting nozzle flow simulation quantities:** The primary objective is to design and train a GNN model, that serves as a surrogate to accurately predict the steady-state velocity and pressure fields of nozzle flow simulations from early, transition states. This model leverages short-term, less computationally intensive simulation results to accurately forecast stable, steady-state flow conditions.

2. **Investigate the accuracy, efficiency and feasibility of the surrogate model:** This research seeks to assess the viability of the developed GNN model as a surrogate to traditional CFD simulations for nozzle flow analysis. The surrogate model's efficiency, in terms of computational resources and time, will be evaluated against conventional simulation methods to establish its practicality for real-world applications. This includes an examination of the model's accuracy as well as a detailed comparison between the steady-state solutions predicted by the GNN models and those obtained from traditional CFD simulations, highlighting the computational savings and potential limitations.

3. **Perform clustering on low-dimensional data to classify simulations:** Another aspect of this thesis is the application of clustering techniques to low-dimensional representations of simulation data. The goal is to categorize simulations based on the velocity ratios between the two inlets of the nozzle to determine occurrences of complete adhesion of the outflow jet to the Coanda surface (referred to in this work as Coanda adhesion) and identify to which wall the complete adhesion takes place. This objective aims to provide deeper insights into the simulation outcomes, facilitating more effective analysis and optimization of nozzle designs and flow conditions.

4. **Investigation of advanced GNN architectures for enhanced model performance**: To explore the potential benefits of incorporating advanced GNN architectures and training strategies, aiming to optimize the model's performance for the complex task of predicting fluid dynamics in nozzle flow scenarios.

Together, these objectives aim to substantially contribute to the field of turbulence modelling using DL techniques, guiding future research and application developments in nozzle flow dynamics and beyond. They offer efficient and adaptable surrogate modelling approach for simulating the dynamics of nozzle flows, thereby reducing the dependence on extensive computational resources and time.

Part II.

Theoretical background - Fluid dynamics and deep learning

2. Nozzle simulation and fluid mechanics primer

This chapter delves into the nozzle flow dynamics of a High-speed Orienting Momentum with Enhanced Reversibility (HOMER) nozzle, developed by Trancossi and Dumas [45]. We begin with an understanding of the problem and the underlying principle, which is the Coanda effect. In Section 2.2, we elucidate the governing equations that mathematically define the fluid flow, as well as additional equations for turbulence modelling. Subsequently, we discuss various numerical approaches that discretize the governing equations. Additionally, the chapter provides insights into the simulation setup in Section 2.4, encompassing meshing strategies, boundary condition specifications, and solver settings for the problem.

2.1. Jet deflection in the HOMER nozzle

The HOMER nozzle is designed to produce a controllable and selective deviation of a synthetic jet, generated by mixing two primitive jets. The jet is controlled solely by taking advantage of the Coanda effect and does not require any mechanical parts. The general structure of the nozzle is depicted in Figure 2.1. As we can see from the figure, the nozzle has two inlets fed by two impinging jets, followed by a convergence zone, or a septum, where mixing of the flows occurs. This generates a synthetic outflow jet, which can be controlled by modifying the momentums of the primitive jets. Next to the convergence zone is the outflow mouth, with curved walls connected to two convex Coanda surfaces on the top and bottom. The system requires a minimum operating condition of the primitive jets [45] to ensure effective mixing. The impinging jets must have velocities high enough to generate a synthetic jet of Reynolds number greater than 5000 at the outlet mouth. To guarantee optimum operation, the Reynolds number at the outlet must exceed 10000. In the case of lower Reynolds numbers, the system's behavior is unpredictable.

2.1.1. Coanda effect

Coanda effect is the tendency of a stream of fluid emerging from an orifice to follow an adjacent flat or curved surface and to entrain fluid from the surroundings so that a region of lower pressure develops. In simple terms, it is the tendency of a fluid to adhere to and stay attached to the walls of a convex surface, as demonstrated in Figure 2.2. Different fluid dynamic effects concur to create the Coanda effect, namely the boundary layer effect,

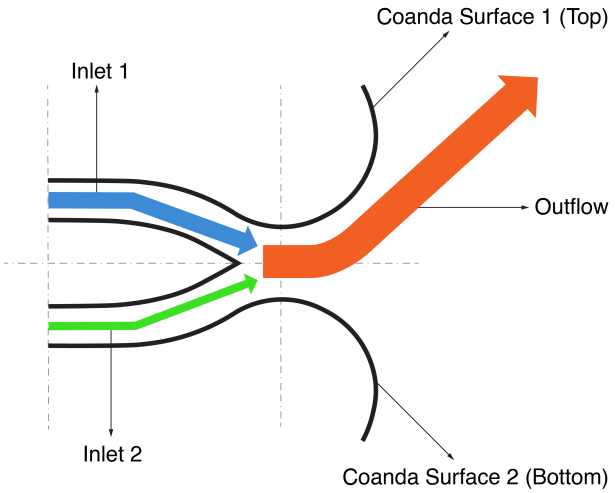


Figure 2.1.: Schematic overview of the HOMER nozzle design, highlighting the dual inlets and Coanda effect surfaces, adapted from Trancossi and Dumas [45]

the adhesion effect, and the attraction effect. Newman [30] has demonstrated that Coanda adhesion to a curved surface is dependent on the equilibrium of forces applied on the fluid. Adhesive motion on a curved surface involves centrifugal force and radial pressure, with contact pressure decreasing due to viscous drag upon jet exit. This pressure differential propels fluid along the curved surface until surface pressure matches ambient pressure, causing detachment between the wall and the fluid jet.

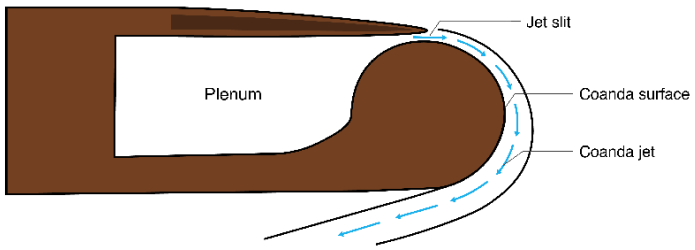


Figure 2.2.: Demonstration of the Coanda effect: Visualization of a fluid jet adhering to and flowing along a curved surface, illustrating the fundamental principle utilized in the HOMER nozzle design.

2.2. Governing equations

In this section, we talk about the mathematical equations that govern the fluid flow in the nozzle setup. The computational domain for our fluid flow problem is shown in Figure 2.3. We consider the same homogenous fluid for both primitive jets. This refers to streams with the same chemical and physical properties, i.e.; the density of the fluid ρ remains constant. The fluid in consideration is air in ideal gas conditions. The Navier Stokes Equations (NSE) can then be used to mathematically model the flow of an incompressible Newtonian fluid within the computational domain, given as,

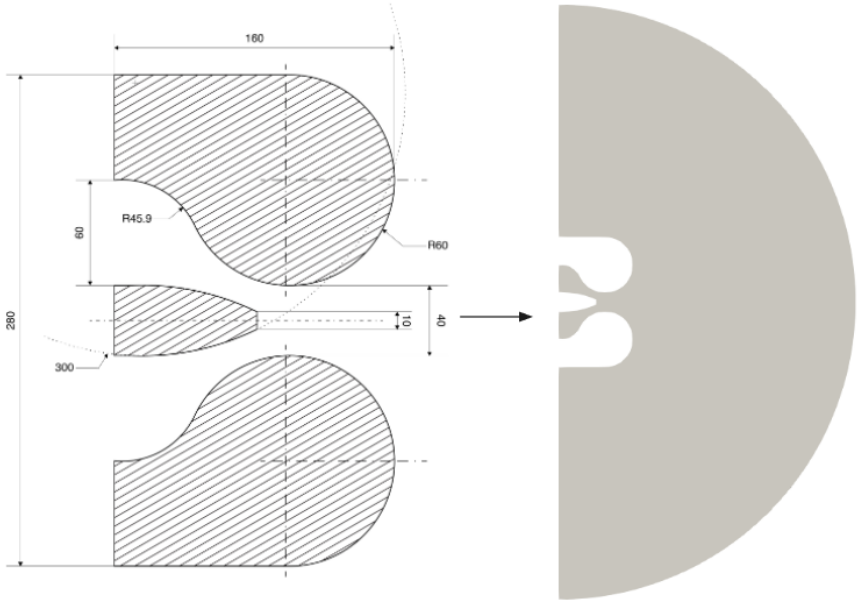


Figure 2.3.: Visualization of the HOMER Nozzle: Left - Geometry of the modified HOMER nozzle; Right - CAD model showcasing the computational simulation domain.

$$\begin{aligned} \frac{\partial u_i}{\partial x_i} &= 0 \\ \frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} &= \frac{-1}{\rho} \frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j^2} \end{aligned} \tag{2.1}$$

where, u_i is the flow velocity in the spatial direction x_i, ν is the kinematic viscosity, μ is the dynamic viscosity ($\nu = \mu/\rho$), and p is the pressure. We set constant velocities at the two inlets based on which a velocity profile corresponding to a fully-developed turbulent plane channel flow is computed at the inlet channels $\partial\Omega_{in}$. At the walls $\partial\Omega_{wall}$, we apply no-slip boundary conditions. We impose zero-gradient Neumann boundary conditions on

the flow quantities at the outlet.
Turbulence, characterized by its unsteady, highly irregular, and energy-dissipative behaviors at high Reynolds numbers, causes minute fluctuations in velocity, pressure, and temperature across varying scales. While a DNS could theoretically capture these fluctuations by solving the NSE, the immense computational resources required render it impractical for most engineering simulations. Turbulence modelling using Reynolds Averaged Navier-Stokes (RANS) equations offers a practical compromise by solving time-averaged equations for steady-state or unsteady RANS (URANS) flows. RANS relies on turbulence models to account for unresolved turbulence effects, allowing for efficient simulations of complex engineering systems without resolving every turbulent detail. The underlying principle is to consider the flow as the sum of the mean flow and turbulent/fluctuating components. For a steady-state flow field, Reynolds decomposition is applied to flow quantities. For example, the flow velocity is expressed as $u_i = \bar{u}_i + u_i'$, where $\bar{u}_i$ is the mean velocity and u_i' is the fluctuating turbulent component. The Reynolds averaging process introduces an additional term to the NSE known as Reynolds stress. By substituting the averaged quantities in the NSE, we obtain the RANS equations for our steady-state, 2D incompressible flow as,

$$\begin{aligned} \frac{\partial \bar{u}_i}{\partial x_i} &= 0 \\ \rho \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} &= -\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left[\mu\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right) - \rho\overline{u_i' u_j'}\right] \end{aligned} \tag{2.2}$$

where, $-\rho\overline{u_i' u_j'}$ is the Reynolds stress tensor term and represents the effect of the small-scale turbulence on the average flow and $\bar{p}$ is the averaged pressure. The RANS equations have no unique solution because they are not in closed form, that is the unknowns are more than the equations. Thus, additional equations are needed for turbulence closure. The most common strategy used in CFD is to relate the Reynolds stress to the shear rate by the Boussinesq relationship:

$$\overline{u_i' u_j'} = -\nu_t\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right) + \frac{2}{3}k\delta_{ij} \tag{2.3}$$

where, k is the turbulent kinetic energy and δ_{ij} is the Kronecker delta, which is 1 if $i = j$ and 0 otherwise. ν_t is the turbulent viscosity, computed from the turbulence models. Some of the RANS-based turbulence models are outlined below:

1. The Spalart-Allmaras model is a computationally efficient, one-equation model that solves for a single variable ν_t.

2. The $k-\epsilon$ model resolves turbulence through two equations: one for turbulent kinetic energy k and another for the rate of dissipation of turbulent kinetic energy ϵ.

3. The $k-\omega$ model is another two-equation model that features transport equations for k and a specific rate of turbulence dissipation ω. This model is particularly advantageous in accurately predicting near-wall flows and is less susceptible to numerical issues than the $k-\epsilon$ model in adverse pressure gradient regions.

4. The $k-\omega$ Shear Stress Transport (SST) model combines aspects of the $k-\epsilon$ model near walls and the $k-\omega$ model away from walls to provide accurate predictions in both regions. The $k-\omega$ SST model is particularly suitable for boundary layer flows, capturing the near-wall behavior accurately while providing robust predictions in the outer flow regions. Its versatility and computational efficiency make it a popular choice for a wide range of engineering applications.

For the purposes of this work, the $k-\omega$ SST turbulence model has been adopted.

2.3. Numerical analysis

Numerical analysis on PDEs - in our case, the RANS equations, is performed by discretizing the continuous domain into a discrete setup. This results in a system of algebraic equations, usually linear systems, that can be solved by iterative techniques such as Jacobi or Gauss-Seidel. Multigrid methods are another class of iterative numerical techniques used to solve discretized PDEs by employing a hierarchy of grids with varying levels of resolution - from coarse to fine - to accelerate convergence.
Some commonly used discretization techniques are the Finite Difference Method (FDM), Finite Element Method (FEM), and Finite Volume Method (FVM). All three methods end up solving one (or several) system(s) of linear equations to compute an approximate numerical solution to the PDE at hand. And for all three methods, these linear systems are sparse, and the equation for an unknown u_i involves a few neighbors of point i. Overall, FDM is mostly used for geometries that can be discretized by structured grids (e.g., rectangles), while FEM and FVM are more suitable for complex domains.
FVM discretizes PDEs by dividing the computational domain into finite volumes or cells. It conservatively approximates integral forms of conservation laws within each cell. The method calculates fluxes across cell interfaces, preserving conservation principles, making it particularly suited for problems involving fluid flow, heat transfer, and other conservation phenomena. As FVM is based on the integral formulation of a conservation law, it is mainly used to solve PDEs in fluid dynamics, which involve fluxes of the conserved variable. In this thesis, we are only interested in the discretization of PDEs using FVM, which is the most widely used discretization approach in CFD solvers.

2.4. Simulation setup

Trancossi and Dumas [45] proposed a mathematical model of the HOMER nozzle and carried out 2D, incompressible flow simulations on this geometry. The simplified model pre-

dicts the detachment angle of the jet stream over the curved surface. The nozzle chosen for our study is a slightly modified version of a thrust-vectoring propulsive HOMER nozzle. The modified design is inspired by the numerical investigation and experimental validation of Kara and Erpulat [19]. The geometry adopted for the simulations as well as the computational domain are depicted in Figure 2.3. The selected channel length ensures the mean flow quantities are fully developed, hence establishing steady-state conditions. The meshing and simulation are carried out on `OpenFOAM` which uses finite volume methods to discretize the PDEs.

2.4.1. Mesh generation

The geometry is created using FreeCAD [11] and patch names are assigned based on the type of boundaries. The meshing process on `OpenFOAM` begins with the discretization of the geometry into hexahedral blocks using `blockMesh`. Then, `snappyHexMesh` refines the mesh based on parameters specified in `snappyHexMeshDict`. This includes defining refinement controls, snapping settings, adding boundary layers, and ensuring mesh quality. The process iteratively refines the mesh until the desired quality and resolution are achieved, enabling accurate simulations of the geometry's physical behavior. An unstructured, 3D hybrid mesh with tetrahedral and hexahedral elements has been generated for the computational domain and is enhanced by boundary layer refinement and a refinement box around the nozzle region.

2.4.2. Boundary conditions and solver settings

We carry out an incompressible steady-state CFD simulation on the mesh, setting appropriate boundary and initial conditions. By applying low Reynolds number wall functions for k, ω, and turbulent viscosity ν_t at walls, we account for near-wall turbulence effects, ensuring accurate modelling near boundaries. We also prescribe adiabatic stationary walls with no slip conditions, and pressure to zero gradient, at the Coanda surfaces and the inner walls of the nozzle. We define the flow entering the domain by prescribing inlets with fixed velocities and set both inlet turbulence intensities to 1% (medium turbulence). Furthermore, we specify the pressure at outlets through a pressure outlet boundary condition, allowing flow to exit the domain without reflecting back. These actions collectively ensure the proper representation of flow behavior within the computational domain. The `simpleFoam` solver is the tool we use for the simulations, employing the Semi-Implicit Method for Pressure Linked Equations (SIMPLE) algorithm for effective pressure-velocity coupling. We iteratively resolve the momentum and pressure equations until achieving a predetermined convergence criterion. The $k-\omega$ SST turbulence model serves our purposes, with the fluid's kinematic viscosity set at 1.51×10^{-5}. For velocity's convective term, we use the linear Upwind scheme for discretization, while the divergence term undergoes discretization via a bounded Gauss linear scheme. For the turbulence fields k, ω, ν_t, we

employ a bounded Gauss upwind scheme for discretization. We have set the convergence criterion to 10^{-6} for the pressure field and 10^{-8} for all other quantities.

2.5. Simulation results

In this section, we highlight the simulation results of a few flow scenarios. Figure 2.4 showcases the velocity fields of four distinct cases and their behavior towards the Coanda surfaces. Figure 2.5 presents the pressure fields for the same cases.

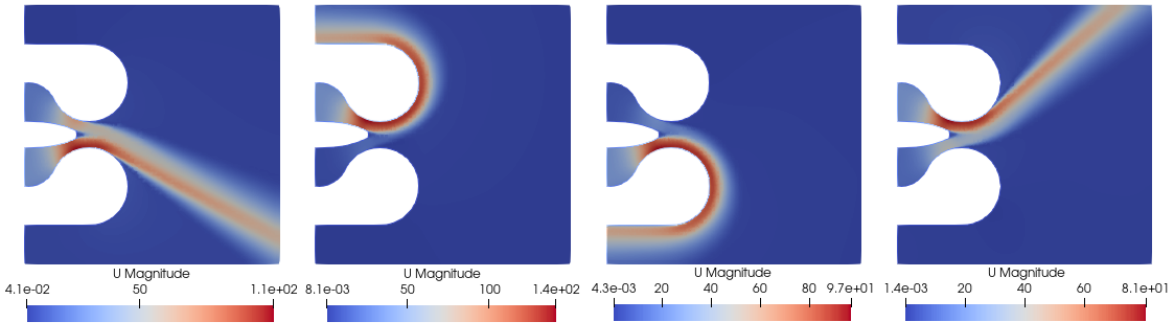


Figure 2.4.: Velocity fields from four simulation cases, from left to right: (a) (18 m/s, 27 m/s) shows jet deflection towards the bottom surface, (b) (35 m/s, 5 m/s) shows Coanda adhesion to the top surface, (c) (6 m/s, 24 m/s) shows Coanda adhesion to the bottom surface, (d) (20 m/s, 10 m/s) shows jet deflection towards the top surface, where (Inlet 1, Inlet 2) describe the initial velocities.

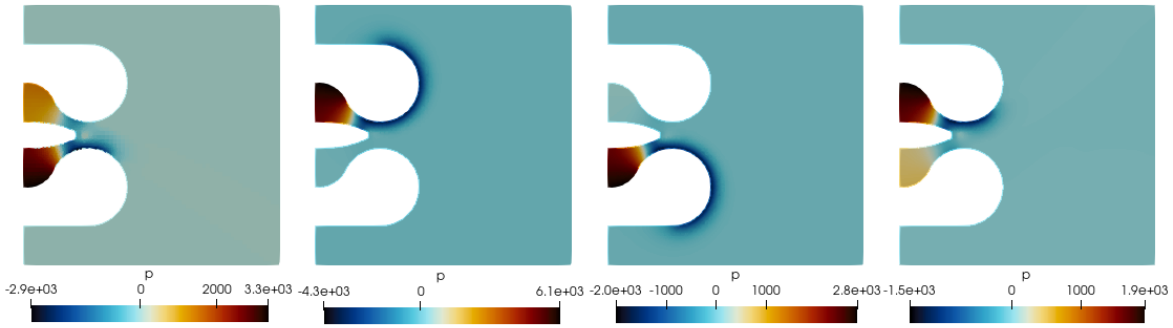


Figure 2.5.: Results from simulations showcasing pressure fields for four cases of interest, from left to right: (a) (18 m/s, 27 m/s), (b) (35 m/s, 5 m/s), (c) (6 m/s, 24 m/s), and (d) (20 m/s, 10 m/s). The initial velocities are enclosed as (Inlet 1, Inlet 2).

3. Deep learning primer

This chapter provides a comprehensive explanation of foundational concepts and methodologies in deep learning, with a particular emphasis on Graph Neural Networks (GNNs). It begins by introducing deep learning and elucidating the basics of neural networks, focusing on their architecture and operational principles. Section 3.3 navigates through the intricacies of training neural networks with topics including feature scaling, weights initialization, batch training, and hyperparameter tuning. Subsequently, it delves into optimization techniques, addressing essential elements such as loss functions, backpropagation, learning rates, and optimizers. Furthermore, this chapter examines model evaluation metrics and explores advanced neural network architectures such as CNNs in Section 3.5 and GNNs in Section 3.6, highlighting their important features and components.

3.1. Introduction to machine learning and deep learning

Machine learning, a dynamic subset of artificial intelligence, is dedicated to developing algorithms that extract insights and patterns from data. This enables systems to enhance their accuracy and decision-making capabilities without being explicitly programmed for each task. The learning process utilizes statistical models and optimization algorithms to iteratively adjust parameters and improve performance. Machine learning approaches are broadly categorized into supervised learning and unsupervised learning based on learning objectives.

Supervised learning involves training a model using labeled data, where each input is paired with a corresponding output. During training, the model learns to map input data to output labels by minimizing the difference between its predictions and the true labels. This approach is commonly used for tasks such as classification and regression.

Unsupervised learning involves training a model on unlabeled data, where the algorithm aims to find hidden patterns or structures within the data without explicit guidance. It is often used for tasks like anomaly detection, data exploration, and feature learning, where the data lacks labeled examples or where the underlying structure is unknown.

Deep learning is a subset of machine learning that has gained significant attention due to its remarkable performance in various tasks, ranging from image and speech recognition to autonomous driving. It utilizes neural networks consisting of multiple layers of interconnected neurons to learn representations of data through iterative processing of input data to make predictions or decisions.

3.2. Fundamentals of neural networks

Artificial Neural Networks (ANN) are computational models inspired by the structure and function of biological neural networks [39]. They consist of interconnected nodes organized into layers, typically including an input layer, one or more hidden layers, and an output layer.

A **perceptron** or an artificial neuron, is the fundamental building block of ANNs. It takes multiple input signals $(x_1, x_2 ... x_n)$, each weighted by a connection weight $w_1, w_2, ... w_n$, sums them up, and applies an activation function σ to produce an output $y = \sigma\left(\mathbf{w}^T\mathbf{x} + b\right)$, where b is the bias, $\mathbf{w}$ and $\mathbf{x}$ are the weight and input vectors, respectively. Perceptrons are arranged in layers to build complex neural network architectures.

Activation functions are usually non-linear functions to introduce non-linearity within

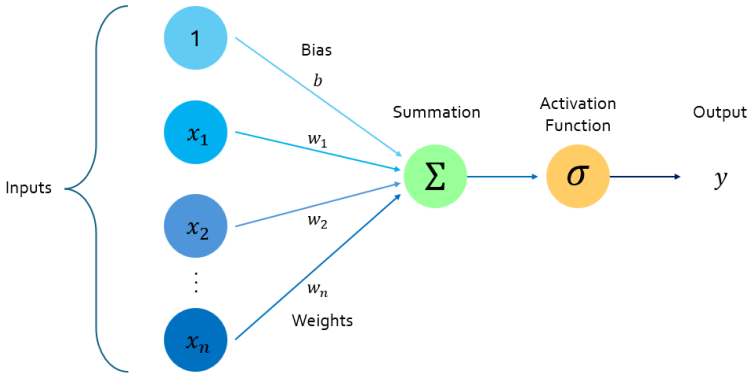


Figure 3.1.: Schematic representation of a perceptron - the basic computational unit of artificial neural networks, illustrating input connections, weights, bias, and an activation function.

the layers of neural networks, allowing them to learn and represent complex relationships in data. Common activation functions include sigmoid, tanh, and Rectified Linear Unit (ReLU) and these are plotted in Figure 3.2. Without the nonlinear transformation via the activation function, the network would be confined to solving only linear problems.

Weights in a neural network represent the strength of connections between neurons. They are learned parameters to adjust the influence of input signals on the neuron's output. **Biases** allow neural networks to model the offset from zero output, influencing the activation of neurons regardless of the input.

Neural Networks (NN) consist of interconnected layers of perceptrons that process input data to produce output predictions. They can be represented as directed graphs, where nodes correspond to perceptrons and edges depict connections between them. These connections typically carry weighted signals from one neuron's output to another neuron's input. In particular, we are interested in **feed-forward neural networks**, in which information flows only in one direction from the input layer through one or more hidden layers

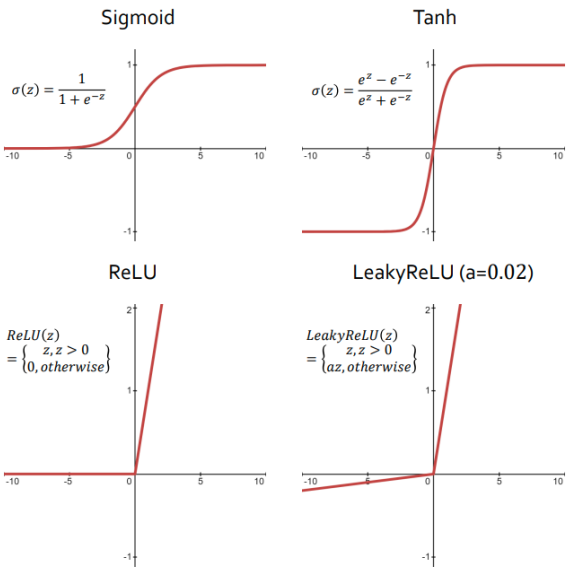


Figure 3.2.: Comparative visualization of common activation functions used in neural networks: (a) Sigmoid, (b) Tanh, (c) ReLU, and (d) Leaky ReLU.

to the output layer. Each layer processes the input data independently, and the output of one layer serves as the input to the next layer. The connections between neurons do not form directed cycles, ensuring that the network architecture is acyclic. Multi Layer Perceptrons (MLP) are the simplest feed-forward neural networks, described in Figure 3.3.

3.3. Training of neural networks

In this section, we delve into the training practices of neural networks, highlighting the optimization of model parameters to enhance task performance. We particularly focus on the methodologies of data handling like data partitioning, feature scaling, regularization techniques, batch training and weight initialization. The process of training neural networks is aimed at optimizing a model's parameters to improve its performance on given tasks. We also talk about optimization strategies, which leverage algorithms like backpropagation to calculate gradients and apply updates via optimizers such as SGD or Adam.

3.3.1. Data partitioning

Data partitioning is a crucial step in deep learning in which the dataset is divided into separate subsets for training, validation, and testing. The data partitioned into training data is used to train the model, and validation data is used to tune hyperparameters and

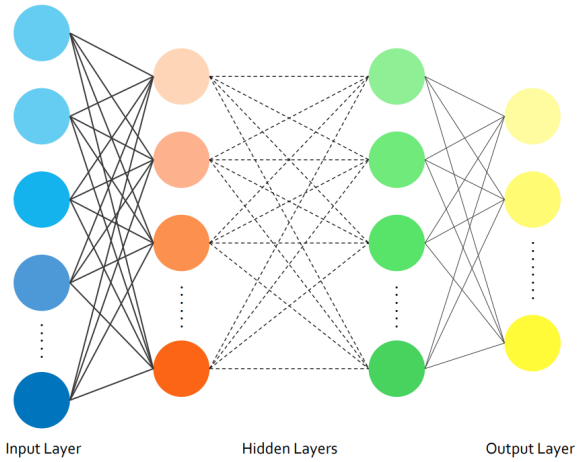


Figure 3.3.: Architecture of a Multi-Layer Perceptron (MLP) showcasing an input layer, multiple hidden layers, and an output layer, demonstrating the flow of information in feed-forward neural networks.

monitor performance. The test data remains unknown during training and is used to evaluate the final model's generalization performance. The partitioning process ensures that the model's performance is assessed accurately on unseen data and provides independent datasets for training and evaluation.

3.3.2. Feature scaling

Feature scaling or normalization, is a preprocessing step aimed at bringing all input features to a similar scale. Features with large magnitudes can lead to large gradients during training, which may cause unstable behavior. Feature scaling mitigates this issue by reducing the range of feature values, thus preventing gradient instability and ensuring more reliable optimization. Common feature scaling techniques include min-max normalization, Z-score standardization, and unit length scaling.

3.3.3. Weight initialization

Weight initialization sets the initial values for the model's parameters before training begins. The initial weights dictate the local minimum the weights should converge to; thus, better initialization leads to improved model performance. Random initialization with small weights is a common practice in deep learning. Initializing weights to random values drawn from a suitable distribution with a zero mean and small variance breaks symmetry and helps prevent both vanishing and exploding gradients. It encourages each neuron to learn different features from the input data, promoting diverse representations

and effective learning. Techniques like Xavier [12] and Kaiming [17] initialization refine this process by adapting to the characteristics of activation functions.

3.3.4. Regularization

Regularization broadly refers to techniques used to prevent overfitting by imposing additional constraints on the model's parameters, i.e.; by adding penalties to the loss function. Regularization penalizes large weights in the model, thereby promoting simpler models that generalize better to unseen data. L_1 regularization encourages sparsity in the weights, performing feature selection by setting irrelevant weights to zero, making the model simpler and more interpretable.

$$\text{Loss}_{\text{L}_1\text{Reg}} = \text{Loss}_{\text{original}} + \lambda \sum_{i=1}^{n} |w_i|$$

L_2 regularization encourages the weights to be spread out more evenly, preventing individual weights from becoming too large.

$$\text{Loss}_{\text{L}_2\text{Reg}} = \text{Loss}_{\text{original}} + \lambda \sum_{i=1}^{n} w_i^2$$

Here, λ is the regularization strength and determines the degree of penalty imposed on large weights. A smaller λ value results in weaker regularization, increasing the risk of overfitting. Conversely, a larger λ value results in a simpler model that generalizes better but may underfit the training data.

3.3.5. Batch training and batch normalization

Batch training is a technique in deep learning in which the model updates its parameters based on a subset (or batch) of the training data, rather than the entire dataset. The training data is divided into batches of fixed size, and the model computes the gradient updates based on an average of the samples in each batch. This reduces the variance in the gradient estimates, stabilizes the gradients, and prevents large fluctuations during training.
Too small a batch size leads to frequent and noisy gradient updates. On the other hand, too large a batch requires more computational resources and memory, despite providing precise gradient estimates and stable optimization. Selecting an optimal batch size requires balancing the trade-off between computational efficiency and the quality of gradients.
Another important term in this context is an epoch, which refers to a single pass through the entire training dataset. During one epoch, each batch is processed sequentially through the neural network, completing a full iteration over the entire dataset. Typically, training iterates over the entire dataset for multiple epochs until the model converges or a predefined stopping criterion is met.

3.3.6. Overfitting and underfitting

Overfitting and underfitting are two common phenomena that affect the performance and generalization ability of the model. Overfitting occurs when a model learns to perform well on the training data but fails to generalize to unseen data. The model becomes overly complex and specific to the training set, leading to poor generalization. Signs of overfitting include high training accuracy but low validation or test accuracy as the model memorizes training examples. Techniques such as regularization, dropout and early stopping can help prevent overfitting by reducing the model's capacity and complexity.
Underfitting occurs when a model is too simple to capture the underlying structure of the data. In this case, the model fails to learn the patterns present in the training data and performs poorly both on the training and unseen data. Underfitting often occurs when the model is too shallow or simple. Signs of underfitting include low training and validation accuracy. Increasing the model's capacity, adding more data, or improving feature engineering can help alleviate underfitting by allowing the model to capture more complex relationships in the data.

3.3.7. Hyperparameters

Hyperparameters in deep learning are fixed parameters set prior to the training process that are not learned from the data. They control various aspects of the learning process, such as the model architecture, optimization settings and the training procedure itself. Some important hyperparameters are the number of neurons per layer, number of layers, activation function, batch size, number of epochs, optimizer, loss function, weight initialization, dropout rate, and regularization strength.
Hyperparameter tuning is the process of selecting the optimal values for these hyperparameters to maximize the performance of the model on unseen data. It involves systematically searching through a predefined space of hyperparameters and evaluating the model's performance using a validation set or cross-validation. The goal is to find the hyperparameters that result in the best generalization performance, balancing between underfitting and overfitting.

k-Fold cross-validation

In k-fold cross-validation [15], the dataset is divided into k subsets or folds, of approximately equal size. The model is trained k times, each time using k-1 folds for training and the remaining fold for validation. This process is repeated k times, with each fold used exactly once as the validation set. Instead of relying on a single validation set, this method averages the performance over multiple folds, providing a more stable estimate of the model's performance. k-fold cross-validation helps evaluate the performance of different hyperparameter configurations during hyperparameter tuning. Figure 3.4 represents Leave One Out Cross-Validation (LOOCV) - a variant of k-fold cross-validation.

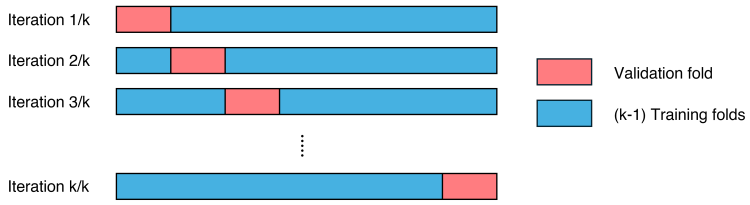


Figure 3.4.: Schematic Representation of LOOCV - In every iteration, exactly one fold is subsequently chosen as the validation fold while the remaining (k-1) folds are combined to become the training dataset.

3.3.8. Optimization

Optimization involves adjusting the parameters of the neural network, such as weights and biases, to minimize a predefined objective function, typically referred to as the loss function. The optimization process iteratively updates the parameters based on the gradients of the loss function with respect to the network's parameters, aiming to converge to a set of optimal parameters that yield the best performance on the given task. Some important aspects of the optimization process are discussed in the following subsections.

Loss function

The loss function quantifies the difference between the model's predictions and the actual target values. It represents the measure of how well the model is performing on the training data. Common loss functions include mean squared error (MSE) for regression tasks and categorical cross-entropy for classification tasks. The loss function $\mathcal{L}(\theta)$ is defined as:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} L\left(y_i, \hat{y}_i; \theta\right) \tag{3.1}$$

Here, θ represents the parameters of the neural network being optimized, such as weights and biases, y_i is the ground truth and $\hat{y}_i$ is the model prediction. The loss function $\mathcal{L}(\theta)$ depends on these parameters, and it is computed as the average of the individual loss $L\left(y_i, \hat{y}_i; \theta\right)$ over all the training examples of size N.

Backpropagation

Backpropagation is a fundamental algorithm used to compute the gradients of the loss function with respect to the parameters (weights and biases) of the neural network. It involves propagating the error backward from the output layer to the input layer, updating the parameters along the way to minimize the loss. The gradients are computed using the chain rule of calculus, enabling efficient optimization of the network's parameters. The

gradients $\nabla_\theta \mathcal{L}(\theta)$ of the loss function are computed as,

$$\nabla_\theta \mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta L\left(y_i, \hat{y}_i; \theta\right) \tag{3.2}$$

Learning rate

The learning rate is a hyperparameter that controls the size of the parameter updates, that is, the step-size in the direction of the gradients computed by backpropagation. A higher learning rate may lead to faster convergence but risks overshooting the optimal solution, while a lower learning rate may result in slower convergence but more stable training. The parameter update rule with learning rate η is given by:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta) \tag{3.3}$$

Here, θ_{t+1} and θ_t represent the parameters at time-step t and $t+1$ respectively. Learning rate decay is often used to gradually reduce the learning rate during training with the help of learning rate schedulers.

Optimizer

The optimizer is responsible for updating the parameters of the neural network based on the gradients computed during backpropagation. It determines the direction and magnitude of parameter updates to minimize the loss function efficiently. Popular optimizers include Stochastic Gradient Descent (SGD) [3], Adaptive Moment Estimation (Adam) [20], Root Mean Squared Propagation (RMSProp) [43], and Adaptive Gradient (AdaGrad) [7]. We use the Adam optimizer in the training and testing phases of our work.

Adam (Adaptive Moment Estimation) is an algorithm for stochastic optimization that combines the ideas of SGD with momentum and RMSProp. It maintains exponentially decaying moving averages of past gradients and past squared gradients for each parameter. These moving averages serve as estimates of the first and second moments of the gradients, respectively. Adam also incorporates bias correction terms to compensate for the initial bias towards zero at the beginning of training. The parameter update rules at time-step t are given by,

1. Compute the gradient of the loss function with respect to the parameters, θ_t.
2. Update biased first moment estimate: $m_t = \beta_1 m_{t-1} + (1 - \beta_1)\theta_t$.
3. Update biased second raw moment estimate: $v_t = \beta_2 v_{t-1} + (1 - \beta_2)\theta_t^2$.
4. Compute bias-corrected first moment estimate: $\hat{m}_t = \frac{m_t}{(1-\beta_1^t)}$.
5. Compute bias-corrected second moment estimate: $\hat{v}_t = \frac{v_t}{(1-\beta_2^t)}$.

6. Update the parameters: $\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{(\sqrt{\hat{v}_t}+\epsilon)}$.

Here, β_1 and β_2 are exponential decay rates for these moment estimates, typically close to 1, and ϵ is a small constant added to avoid division by zero.

3.4. Model evaluation metrics

Model evaluation metrics are essential for assessing the performance of deep learning models. The loss function serves as a crucial metric for evaluating model performance, in addition to optimizing model parameters during training. In regression tasks, where the goal is to predict continuous values, common loss functions include the following:

- **Mean Absolute Error (MAE):** MAE measures the average absolute difference between the predicted values and the actual values:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

  MAE is robust to outliers and does not penalize large errors heavily.

- **Mean Squared Error (MSE):** MSE measures the average squared difference between the predicted values and the actual values:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

  MSE penalizes larger errors more heavily than MAE since errors are squared. This makes it more sensitive to outliers.

- **Root Mean Squared Error (RMSE):** RMSE is the square root of the average squared difference between the predicted values and the actual values:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

  RMSE is sensitive to outliers, similar to MSE, but is more interpretable as it is in the same units as the target variable.

3.5. Convolutional Neural Networks (CNNs)

CNNs [24] are specialized neural networks designed to process grid-like data, such as images. They utilize convolutional layers and pooling operators to learn spatial hierarchies of features from input data, making them highly effective for tasks like image classification, object detection, and image segmentation. In this section, we outline the major components of CNNs, such as convolutional layers, pooling and unpooling operations.

3.5.1. Convolutional layer

The convolutional layer in a CNN consists of a set of learnable filters (kernels) that slide over the input data, performing element-wise multiplication and summing to produce feature maps. The convolution operation preserves the spatial relationship between pixels and learns local patterns like edges, textures, and shapes. Figure 3.5 shows the working of a convolution kernel.

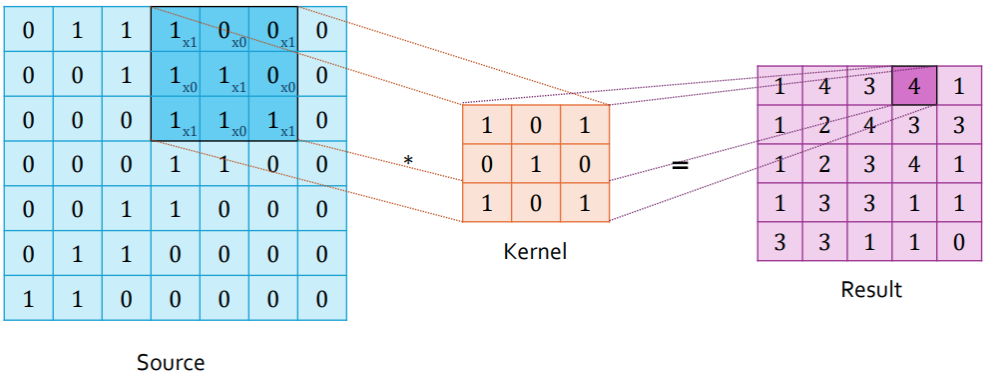


Figure 3.5.: Detailed view of a convolutional layer's operation in a CNN, depicting the convolution process over an input matrix with a specified kernel to produce feature maps.

3.5.2. Pooling and unpooling

Pooling and unpooling operators perform sampling operations, enabling hierarchical feature extraction while preserving spatial information. Pooling is a down-sampling operation commonly used in CNNs to reduce the spatial dimensions of feature maps. Max pooling and average pooling are popular pooling techniques that select the maximum or average value within each pooling region, respectively. These operations are depicted in Figure 3.6. Conversely, unpooling layers, used in upsampling, reconstruct the original input resolution from the lower-dimensional representations generated by pooling. These layers store the indices of the maximum values during pooling and use them for upsampling. Nearest neighbor interpolation is a simple upsampling method where each pixel in the input is replicated multiple times to form the output, as seen in Figure 3.7.

3.5.3. The U-Net architecture

U-Net [38] is a CNN consisting of a U-shaped network structure with a contracting path (encoder) followed by an expanding path (decoder), which is widely used for image seg-

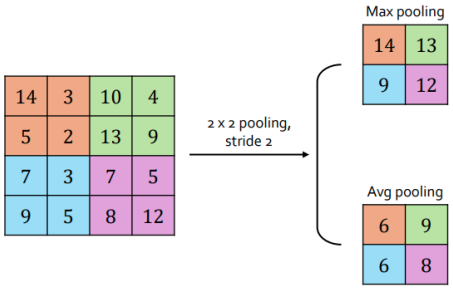


Figure 3.6.: Visualization of pooling operations in CNNs, (a) max pooling, and (b) average pooling.

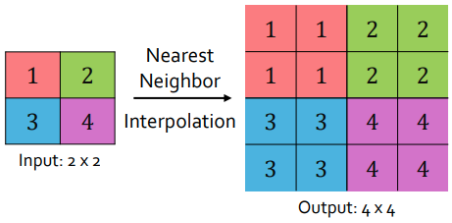


Figure 3.7.: Illustration of nearest neighbor interpolation based unpooling used in CNNs.

mentation. Some important components of the U-Net architecture are discussed below.

1. The **encoder** comprises a series of down-convolutional and max pooling layers that gradually reduce the spatial dimensions of the input image while increasing the number of feature channels. This path extracts high-level features from the input image while preserving spatial context.

2. The **decoder** consists of up-convolutional (transposed convolution) and concatenation layers that gradually increase the spatial dimensions of the feature maps while reducing the number of feature channels. This path generates segmentation masks by upsampling the low-resolution feature maps obtained from the encoder and combining them with high-resolution feature maps using skip connections.

3. **Skip connections** or residual connections [18], are direct connections between layers at the same hierarchical level in the network. Skip connections connect the encoder to the corresponding layers in the decoder. This enables the network to retain fine-grained spatial information from the encoder while recovering spatial details lost during downsampling.

A notable observation pertinent to our current endeavor is the striking resemblance between the U-Net architecture and the V-cycle multi-grid method, as noted by He and Xu

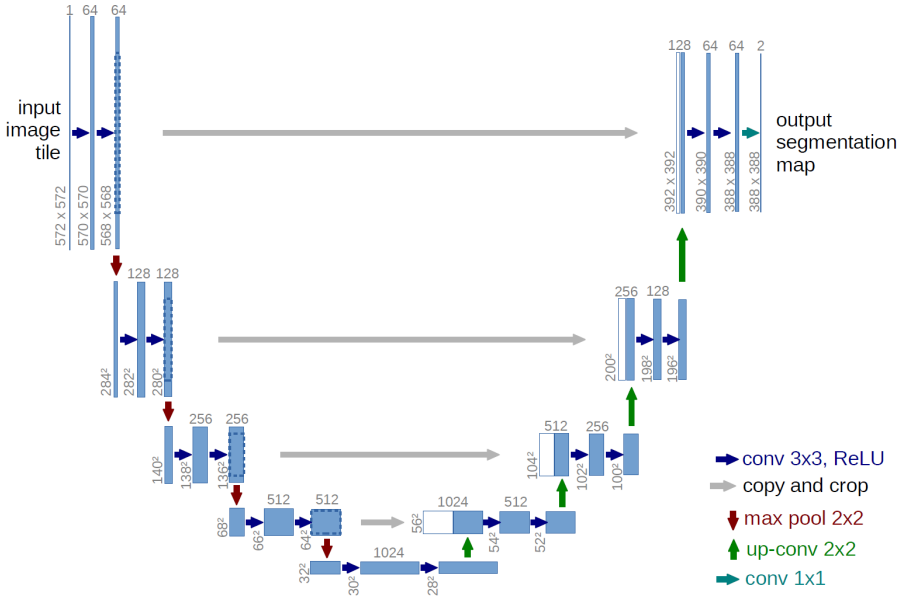


Figure 3.8.: Structure of the U-Net architecture demonstrating its U-shaped design with contracting and expanding paths. Image taken from [38].

[16]. Both employ a hierarchical structure wherein information is exchanged across varying resolutions. A major limitation of CNNs is their inability to directly operate on irregular data formats, such as social networks, recommender systems, molecular structures, or citation networks. In contrast to CNNs, which are well-suited for grid-like structured data such as images, GNNs are tailored for data represented as graphs, which are discussed in the upcoming section.

3.6. Graph Neural Networks (GNNs)

In 2017, Kipf and Welling introduced the Graph Convolutional Network (GCN) [21], a foundational architecture that laid the groundwork for modern Graph Neural Networks (GNNs). Since then, numerous advancements and variants of GNNs have been proposed. Graph data are characterized by non-Euclidean and irregular structures, where entities (nodes) and their relationships (edges) vary in connectivity and structure. GNNs excel in processing unstructured data by leveraging the inherent graph structure by dynamically aggregating information from neighboring nodes based on their connectivity. Formally, a graph G can be denoted as $G = (V, E)$, where V is the set of nodes and E is the edge set. The key components of a graph are outlined below.

1. **Nodes**: Nodes represent entities in a graph, such as users in a social network, atoms in a molecule, or words in a document.

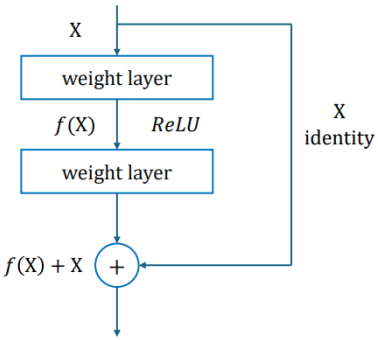


Figure 3.9.: Diagram demonstrating the concept of skip connections within neural network architectures. Skip connections bypass one or more layers by directly feeding the output from an earlier layer to a later layer.

2. **Edges**: Edges define relationships or connections between nodes in a graph. Each edge e_{ij} connects node v_i to node v_j, where $v_i, v_j \in V$. The edge set E can be represented as a collection of tuples (v_i, v_j) indicating the connections between nodes.

3. **Adjacency matrix**: An adjacency matrix is a binary $N \times N$ matrix representing the connections between N nodes in a graph. For an undirected graph, A_{ij} is 1 if there exists an edge between nodes v_i and v_j, and 0 otherwise. For directed graphs, the adjacency matrix may be asymmetric to represent the directionality of edges. The adjacency matrix A of the graph G can be defined as,

$$A_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

4. **Node attributes and node feature matrix**: Node attributes or features represent information associated with each node in the graph. These features can encode characteristics such as velocity, pressure, and temperature as node embeddings. The node feature matrix X for a graph with N nodes and D features is a $N \times D$ matrix where each row corresponds to a node and each column represents a feature dimension, given by,

$$X = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{bmatrix} \quad \text{where,} \quad x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{iD} \end{bmatrix}$$

where x_i represents the feature vector associated with node v_i.

5. **Edge weights**: Edge weights W quantify the strength or intensity of relationships between nodes connected by edges. These weights can represent similarity measures,

distances, or any other relevant information associated with edge connections. Similar to node weights, edge weights can be learned or predefined.

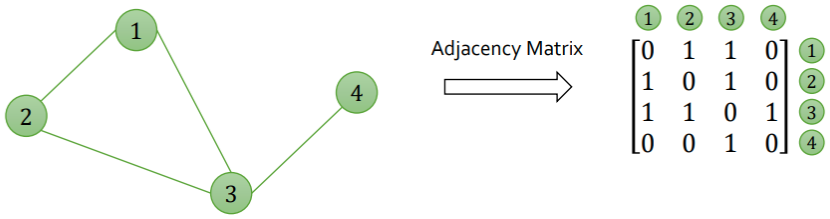


Figure 3.10.: Schematic diagram of graph connectivity depicted by an adjacency matrix using binary representation of node relationships in graph-structured data.

GNNs leverage these components to perform message passing and aggregation operations across the graph structure, which are discussed below.

3.6.1. Graph convolutions

Graph convolutions update the feature representations of nodes in a graph by aggregating information from their neighboring nodes. Graph convolutional operators typically exhibit local connectivity, where each node's representation is updated based on the information from its neighboring nodes. This local connectivity property allows the model to capture localized patterns and dependencies within the graph structure. Weight sharing is a key aspect of graph convolutions, where the same set of learnable parameters (weights) is shared across different nodes in the graph. This allows for parameter efficiency and enables the model to generalize well to unseen nodes and graphs. GCNConv [21], GMMConv [29], and SAGEConv [14] are some common classes of convolution operators. Here, we discuss the GCNConv operator that is used in Graph Convolutional Networks (GCNs).

GCNConv

GCNConv aggregates information from neighboring nodes and updates the representations of each node based on this aggregated information. The main steps involved are:

1. **Message passing**: Nodes exchange messages with their neighbors, aggregating information from neighboring nodes. The message passed from node v_j to node v_i at layer l can be represented as:

$$m_{ij}^{(l)} = \frac{1}{c_{ij}} \mathbf{W}^{(l)} h_j^{(l)} + \mathbf{B}^{(l)} h_i^{(l)}$$

where $m_{ij}^{(l)}$ is the message from node j to node i, $\mathbf{W}^{(l)}$ and $\mathbf{B}^{(l)}$ are the learnable weight and bias matrices, $h_i^{(l)}$ denotes the feature vector of node i at layer l, and c_{ij} is a normalization factor.

2. **Aggregation**:Nodes aggregate the messages received from their neighbors to update their own feature representations. The aggregated message $a_i^{(l)}$ for node i can be computed as the sum or average of the incoming messages.

$$a_i^{(l)} = \sum_{j \in \mathcal{N}(i)} m_{ij}^{(l)}$$

where $\mathcal{N}(i)$ denotes the set of neighboring nodes of v_i.

3. **Update**: Nodes update their feature representations using the aggregated messages and their own features. The updated feature representation $h_i^{(l+1)}$ for node i at layer $l+1$ can be computed as:

$$h_i^{(l+1)} = \sigma(a_i^{(l)}) = \sigma\left(\sum_{j \in \mathcal{N}(i)} \frac{1}{c_{ij}} \mathbf{W}^{(l)} h_j^{(l)} + \mathbf{B}^{(l)} h_i^{(l)}\right)$$

These steps are performed iteratively across multiple layers of the GNN. At each layer, nodes update their feature representations based on the aggregated messages. The iterative propagation of messages allows nodes to incorporate information from distant parts of the graph and refine their representations over multiple layers.

3.6.2. Graph pooling

Graph pooling aggregates node representations across the entire graph to compute global graph-level features and create a coarser graph representation. It reduces the size of the graph representation while preserving important structural and relational information. The different types of graph pooling are:

Top-k pooling

This algorithm selects the top k nodes based on criteria such as node importance or feature values, and aggregates their information to create a coarser graph representation. It retains the most informative nodes while reducing the size of the graph, making it suitable for tasks requiring node selection or summarization. It uses a pooling ratio approach such that the graph has $\lfloor kN \rfloor$ nodes after the pooling operation, where $k \in (0, 1]$ is the pooling ratio. The decision of which nodes to discard is based on a projection score computed against a learnable vector.

Max pooling

Max pooling selects the node with the maximum feature value from each neighborhood and aggregates their information to create a coarser representation of the graph. It emphasizes the most salient nodes in each neighborhood, capturing important features while reducing redundancy.

3.6.3. Graph unpooling

Graph unpooling is a complementary operation to graph pooling, aimed at upsampling or reconstructing the original graph representation after downsampling. While graph pooling creates a coarser representation, graph unpooling aims to recover the finer details and restore the original graph structure. Some common types of graph unpooling include,

Nearest neighbor interpolation

k - Nearest Neighbors (k-NN) interpolation, introduced with PointNet++ [35] for graphs, is a technique used in GNNs to upsample or unpool graph data. In this method, for each node in the coarser graph, the k nearest neighbors from the finer graph are identified. Then, the features of these k-NN nodes are combined to interpolate the features of the target node. Let z be a node from the coarser mesh M_1, and assume its k nearest neighbors on the finer mesh M_2 are denoted as $x_1, \ldots, x_k$. For a node feature f, the interpolated feature $f(z)$ of M_1 is defined based on the features of the k nearest points to z on M_2 as,

$$\mathbf{f}(\mathbf{z}) = \frac{\sum_{i=1}^{k} w\left(x_i\right) \mathbf{f}\left(x_i\right)}{\sum_{i=1}^{k} w\left(x_i\right)}, \text{ where } w\left(x_i\right) = \frac{1}{\left\|z - x_i\right\|_2} \tag{3.4}$$

Max unpooling

A common unpooling strategy used in conjunction with max pooling, the locations of the maximum activations are stored. In max unpooling, these locations serve as masks to place the pooled values back into their original positions in the unpooled feature map.

Part III.

Application and analysis

4. Implementation, results and discussion

This chapter begins with a detailed exploration of the processes and methodologies employed to implement GNNs for the predictive analysis of nozzle flows. In the subsequent section, we introduce the Graph U-Net architecture, outline its implementation settings, present the results obtained, and validate them with the simulation outcomes. This discussion sets the stage for the development of three modified surrogate models with the aim of enhancing model performance, detailed in Section 4.3. Through quantitative analysis and performance evaluation, we identify the successes and challenges encountered in modelling nozzle flow dynamics with GNNs. Thus, this chapter demonstrates the practical implementation of the proposed methodologies and performs a critical examination of the results and the efficacy of these models.

4.1. Data pre-processing

In this section, we describe the process for dataset preparation, how the data from CFD simulations are extracted and converted into graph-structured formats that facilitate efficient GNN training. It includes dataset generation from nozzle flow simulations at varied conditions, transformation of unstructured mesh data into a graph format for GNN compatibility, and the specification of GNN inputs and outputs.

4.1.1. Dataset generation

Nozzle simulations are carried out for 120 cases, each with a different set of Inlet 1 and Inlet 2 velocities. The velocity ratio between the two velocities lies in the range of [1, 10]. Velocity ratio in our case refers to the ratio of higher velocity to lower velocity. Simulation data are obtained at two intervals: after 1000 and 30000 iterations or steps. The results at 1000 iterations, which are still developing and unstable, serve as inputs to the surrogate model. Whereas, the results at 30000 iterations, which represent stable solutions, are considered the ground truth or target data for model training. These results are then transformed into graph data, which encapsulate the spatial relationships and properties of flow fields. The GNN architecture is designed to learn these features and predict stable, steady-state fields from the early, unstable simulation results.

4.1.2. Transformation of mesh data to graph data

Conventional RANS solvers require substantial distances from domain boundaries to mitigate adverse effects on solutions around the region of interest. However, this is not required for the deep learning task. Hence, we narrow our attention to a small region just enclosing the nozzle, as seen in Figure 4.1. We clip the CFD mesh appropriately and resample the velocity and pressure fields to this mesh with reduced spatial extent. We define the cell centers on the clipped mesh and assign them as the nodes of the graph. Two adjacent cells i and j (cells that share an edge) in this resulting unstructured mesh are represented as nodes v_i and v_j on the graph and are connected by an edge e_{ij}. The graph connectivity is then given by the edge index data structure, which comprises two lists - one stores the source node indices, and the other has the destination node indices. CFD solvers typically assign pressure, velocity and other fields to each cell of the mesh, whereas graphs require node features, i.e.; fields defined on each node. Therefore, the cell data (fields) are converted to point data at the cell centers, making them suitable for graph representation. The simulation data is then saved in a `hdf5` format. This is then directly used to read pressure, velocity and co-ordinate information at cell centers. The edge index data required for the GNN is generated by computing adjacent cells and storing their indices in a Co-Ordinate list (COO) format.

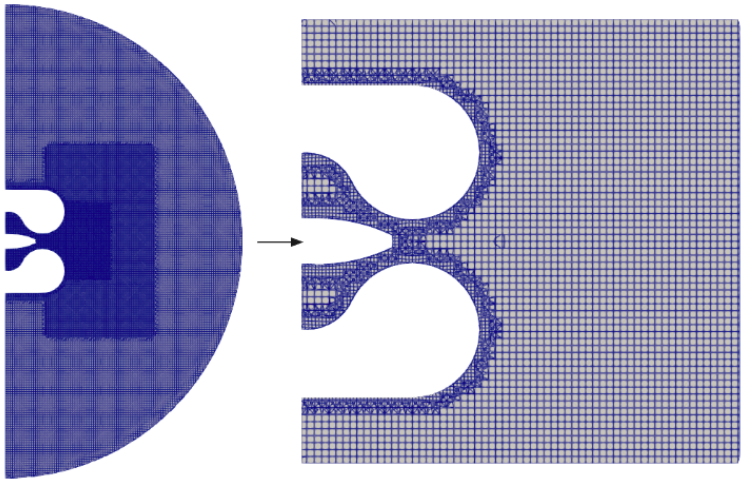


Figure 4.1.: Depiction of the area of focus - the original CFD mesh (left) is clipped and transformed into the region of interest (right)

4.1.3. Model inputs and outputs

After data pre-processing, the simulation mesh is considered as a bidirectional graph $G = (V, E)$ where the set of N nodes denoted as V are linked by the set of edges E of the graph. To construct a graph, we need,

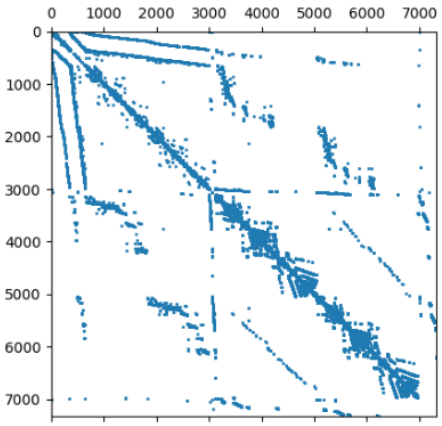


Figure 4.2.: Visualization of the adjacency matrix representing the graph connectivity of the clipped, unstructured mesh with 7329 nodes.

- A feature description, consolidated into an $N \times D$ feature matrix X, where D denotes the number of input node features.
- The graph connectivity or relationships within nodes is represented in matrix form as an adjacency matrix, A or as an edge set E of the shape $2 \times P$, where P is the number of pairs of connected nodes in E.

Let each node have F_X features, and F_Y predictions. The GNN maps the set of node features and edge index matrices to predictions as,

$$\text{GNN} : \mathbb{R}^{N \times F_\text{X}}, \mathbb{W}^{2 \times P} \rightarrow \mathbb{R}^{N \times F_\text{Y}} \tag{4.1}$$

We then get a graph-level output of the shape $N \times F_\text{Y}$. The node feature vector $\mathbf{x}_i$ and prediction vector $\mathbf{y}_i$ of interest at each node v_i is given as,

$$\begin{aligned} \mathbf{x}_i &= [u_{x,i}, u_{y,i}, c_{x,i}, c_{y,i}, \gamma_{\text{tag},i}] \\ \mathbf{y}_i &= [u_{x,i}, u_{y,i}, p_i] \end{aligned} \tag{4.2}$$

where $u_{x,i}$ and $u_{y,i}$ are the node velocities in X and Y directions, p_i is the node pressure, and $c_{x,i}$ and $c_{y,i}$ are the spatial co-ordinates of the nodes. $\gamma_{\text{tag},i}$ is the node tag that defines which cell the node belongs to: inlet, walls or internal mesh. To summarize, our model has 5 input channels (representing node features) and 3 output channels (denoting node predictions). In addition to these channels, the GNN model also requires an edge index matrix to internally compute the adjacency matrix for the graph.

4.1.4. Data normalization

Data normalization is performed on both input channels (node features) and output channels (target vectors), carried out using the three steps outlined below.

1. Following common practice, we normalize all the fields of interest with respect to the magnitude of free-stream or reference velocity u_0 to make them dimensionless.

$$\tilde{u} = u/\left\|u_0\right\|, \quad \tilde{p} = p/\left\|u_0\right\|^2 \tag{4.3}$$

   The latter plays a crucial role as it eliminates the quadratic scaling effect present in the pressure values of the target data, effectively flattening the solution space, thereby simplifying the task for the neural network in subsequent stages.

2. Next, we subtract the mean pressure from the dimensionless pressure values.

$$\hat{p} = \tilde{p} - p_{mean}, \quad \text{where} \quad p_{mean} = \sum_i p_i/n \tag{4.4}$$

   n is the number of training samples and p_i denotes individual pressure values. Without this step, the pressure targets depict an ill-posed learning objective since the random pressure offsets in the solutions lack correlation with the inputs.

3. As a final step, every channel undergoes normalization to the range of [-1, 1] or [0,1]. This standardization aims to mitigate errors stemming from finite numerical precision during the training period. We opt for the maximum absolute value of each quantity across the entire training dataset to normalize the data.

After performing normalization, we shuffle the entire dataset, split it into 3 parts and distribute it as training data, validation data and test data in the ratio of 80:10:10.

4.2. Graph U-Net

Here, we introduce the Graph U-Net architecture, a foundational framework for the surrogate models used in this work. We analyze the benefits and shortcomings of this model as well as explain the motivation behind developing a modified GNN.

4.2.1. Architecture

GCNConv layers are used to perform convolution operations in Graph U-Nets. The downsampling operation is carried out using the top-k pooling strategy, which retains the most important nodes in the graph while discarding less relevant nodes based on a specified criterion. Here, unpooling is not a distinct operation like in traditional U-Nets. Instead, skip connections are used to implicitly perform unpooling. During decoding, downsampled features from the encoder are combined with zeros or empty features in the decoder using skip connections. This integration effectively restores spatial details and contextual information from the original input graph, ensuring that important features are retained and allowing for the recovery of detailed graph structures. Therefore, unpooling in Graph U-Net is seamlessly integrated into the skip connection mechanism, facilitating the reconstruction of the original graph resolution during decoding. Figure 4.3 presents a schematic overview of the Graph U-Net architecture.

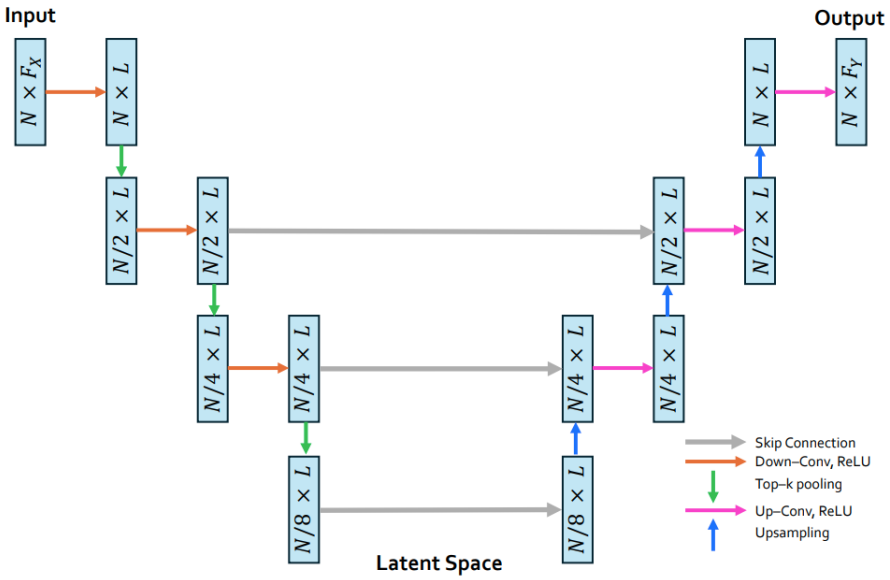


Figure 4.3.: Detailed view of the Graph U-Net architecture, showcasing an encoder-decoder structure with skip connections. Each layer's application is annotated with the resultant dimensions, illustrating the feature reduction or expansion throughout the network. Here, L is the number of channels in a convolutional layer.

4.2.2. Results and discussion

We develop a surrogate model that uses the exact Graph U-Net architecture. As a first step, we are interested in investigating the ability of the model to reproduce the target data, i.e.; when the same target data is given as input. This means that the task solely performs reconstruction of the target dataset without requiring prediction. We also go on to perform the prediction task with the same model architecture, which forms the baseline for all the other architectures proposed in this work. The following settings are maintained for both the reconstruction and prediction tasks. We implement a 9-fold cross-validation for the training process. The initial learning rate is set to 0.0005, and a Step LR scheduler is used to decay the learning rate by a factor of 0.75 after every 100 epochs. We use the Adam optimizer and train on the RMSE loss for 500 epochs. The model's hyperparameters are selected by a hyperparameter tuning process. Table 4.1 presents the model complexity and performance on varying the number of channels and hidden layers. It is to be noted that each hidden layer refers to the combination of sampling (pooling and unpooling) and convolution operations (up and down convolutions). That is, if we perform the pooling and convolutions d times in the encoder and reverse this process d times in the decoder, the number of hidden layers in this architecture is taken as d.

Out of the various architectures, we select the model that gives the lowest training loss

Table 4.1.: Hyperparameter tuning - Table depicting the number of channels, hidden layers, trainable parameters, training loss, and validation loss, measured with the RMSE criterion, for the Baseline architecture corresponding to each setting.

Channels	Hidden layers	Trainable parameters	Training loss	Validation loss
48	2	7k	0.04876	0.05166
	3	12k	0.04713	0.05128
	4	**17k**	**0.04386**	**0.05237**
	5	22k	0.04272	0.05299
64	2	13k	0.04536	0.05349
	3	21k	0.05074	0.05472
	4	30k	0.04139	0.05615
	5	31k	0.04692	0.05587
128	2	51k	0.03983	0.05639
	3	84k	0.03840	0.05492
	4	117k	0.03642	0.05341
	5	150k	0.05011	0.05563

and validation loss, or one that offers a good compromise between the two losses. Among the models with similar performance, we choose the one that is the simplest, i.e.; has relatively feweer trainable parameters, without compromising on accuracy or facing the risk of overfitting. Hence, we execute the training process using a GNN architecture with 48 channels and carry out the sampling and convolution operations 4 times (number of hidden layers) on the encoder and the decoder sides. Similar experiments have been conducted to choose the ideal batch size, activation function and other hyperparameters. The key hyperparameters are listed in Table 4.2.

Table 4.2.: Model hyperparameters

Hyperparameter	Value/Description
Channels	48
Hidden layers	4
Pooling ratios	[0.5,0.5,0.5,0.5]
Batch size	4
Activation function	ReLU
Weight initialization	Kaiming
Initial learning rate	0.0005

Reconstruction task

As mentioned previously, we prescribe the same steady-state CFD dataset as the input and target data to evaluate the effectiveness of reconstruction of the Graph U-Net model. The CFD results, predictions of the GNN model and the absolute difference between them for four simulation cases from the test dataset are shown in Figure 4.4. We note down some

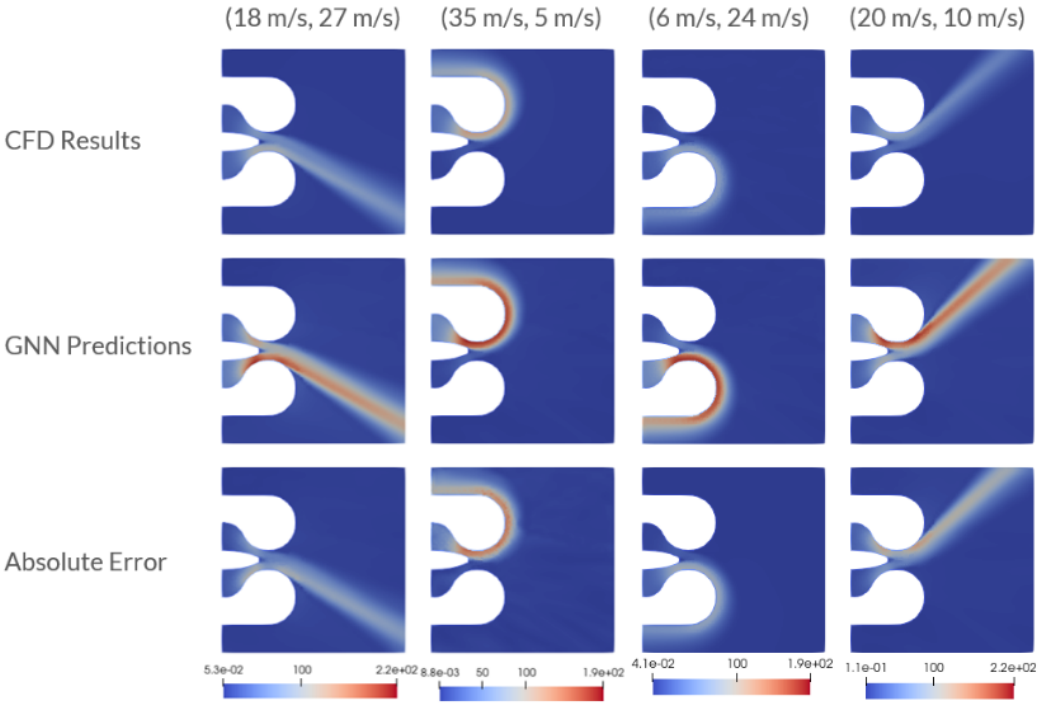


Figure 4.4.: Visualization of velocity fields for four cases from the baseline's reconstruction task, with inlet velocity values prescribed as (Inlet 1, Inlet 2). Here, the first row represents the target data, the second row corresponds to the model predictions, and the last row is the absolute difference between them. The colour bar denotes the magnitude of velocity.

key observations:

1. The GNN predictions closely mimic the overall flow patterns observed in the CFD results across all the different flow conditions, suggesting the model has a good grasp of the flow dynamics around the nozzle.

2. There are discrepancies in the velocity magnitude between the CFD results and GNN predictions. These are observable in the color intensity differences, indicating that the GNN overpredicts the velocity magnitude of the outflow jet. The pressure field, though not visualized here, also faces the same problem.

3. There is a visible discrepancy in the finer details - the GNN seems to smooth out some finer features particularly behind the convergence zone and around the walls of the nozzle.
4. The areas of highest error (red zones) tend to occur in regions with complex flow features, such as sharp gradients, separation and attachment points.
5. In regions where the flow is more predictable and less influenced by the geometry, such as the far field away from the object, the GNN predictions have lower errors.

The scale discrepancy observed could be due to several reasons. Rounding errors in normalized quantities may be more pronounced when converting them back to their original scale for visualization, due to limited floating-point precision. Furthermore, loss of precision for smaller values can occur on undoing the normalization, due to the dynamic range of data (presence of extreme values). The lack of explicit unpooling layers in Graph U-Nets can also be a cause of inadequate reconstruction of the original graph structure during decoding.

Prediction task

In this case, we predict the steady-state solutions from the earlier transition state. The CFD results, predictions of the GNN model, and the absolute difference between these data for four simulation cases from the test dataset are shown in Figure 4.5. To better comprehend and evaluate the model performance, we estimate the training, validation and test losses of the baseline model for both tasks in Table 4.3. We also note down the absolute difference between the input and target data for the test dataset of the prediction task. Some important observations are:

Table 4.3.: Model evaluation metrics of the Baseline model for the reconstruction and prediction tasks.

Model	**Training Loss**	**Validation Loss**	**Test Loss**	**Abs. diff.**
Baseline - Reconstruction	0.025627	0.028391	0.029312	0
Baseline - Prediction	0.1459	0.1503	0.1635	0.2626

1. For the different flow conditions presented, the GNN seems to capture the correct flow patterns, such as the direction of the flow and areas of flow separation and reattachment around the geometrical feature.
2. There is a clear mismatch in the velocity magnitude between the GNN predictions and the CFD results, indicated by the difference in their colour intensities.
3. The loss values suggest that the model is generalizing reasonably well, although there's a slight increase in the test loss, indicating potential overfitting or high model variance when exposed to new data.

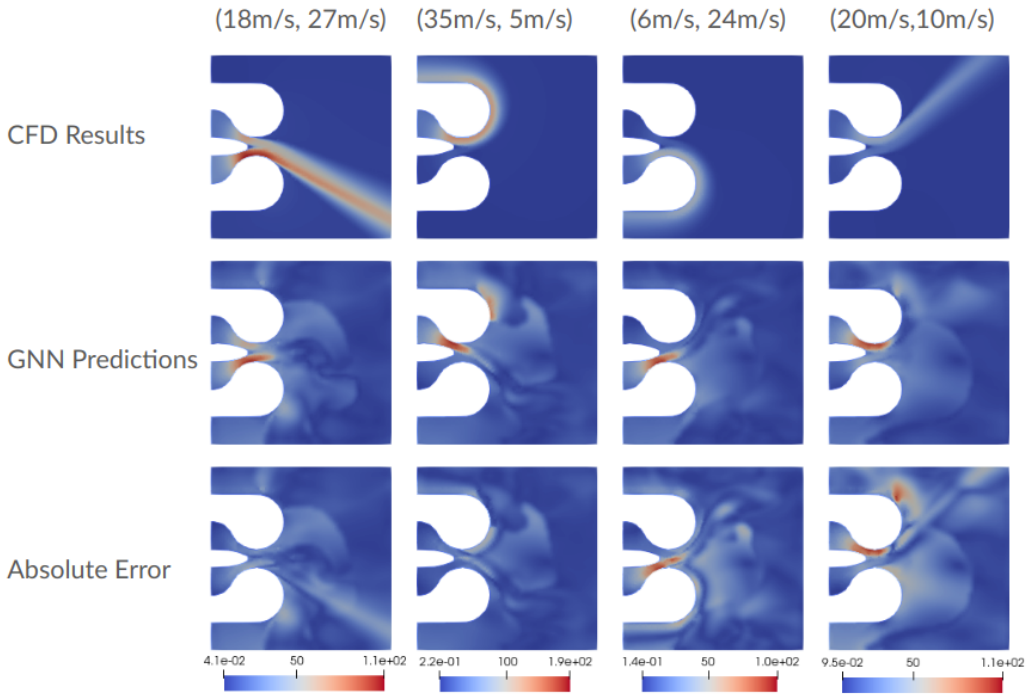


Figure 4.5.: Visualization of velocity fields for four cases from the baseline's prediction task, with inlet velocity values prescribed as (Inlet 1, Inlet 2) presented in four columns. Here, the first row represents results the target data, the second row corresponds to the GNN predictions for the velocity field, and the last row is the absolute difference between the target data and GNN predictions. The colour bar denotes the magnitude of velocity.

4. The mismatch is especially pronounced in regions where velocity gradients are significant, such as the flow constriction and separation regions.

5. Even in the far-field, the GNN model does not seem to replicate the exact velocity magnitudes, as observed from the consistent overestimation of velocity magnitude across different flow regions.

Thus, the model generally captures the flow patterns accurately across various conditions, as shown by the similarity in the large-scale flow features in the CFD and GNN results. The model has learned to approximate the fluid dynamics involved with a fairly good degree of accuracy given the proximity of the training, validation, and test losses.

4.2.3. Limitations

Originally designed for small graphs with around 100 nodes, Graph U-Nets relies on dense matrix multiplications, which are memory-intensive and not scalable. This leads to memory constraints and slower training times, thus making it impractical for complex, large-scale graph data. GCNConv layers used in this architecture may have limited expressiveness in capturing higher-order graph structures and long-range dependencies in the graph. Its optimal depth is found to be 2 or 3 layers [21]. Deeper models beyond 7 layers can encounter training difficulties due to the risk of overfitting. Apart from these issues, there is also a lack of a dedicated upsampling layer in the architecture, which leads to poor reconstruction of the latent space. Due to these disadvantages, Graph U-Net may exhibit poor performance in terms of both accuracy and efficiency, particularly for complex geometries or large datasets. Hence, there is a paramount necessity to rely on modified GNN architectures for our work.

4.3. Proposed architecture

In this section, we propose three GNN surrogate models and elucidate the architecture and improvements made to the original Graph U-Net framework. Then, we proceed to provide details on the hyperparameters and other implementation specifics of the proposed models. Finally, we demonstrate the training process and share the prediction results obtained for the CFD application. The models are developed on the Pytorch deep learning framework using the Pytorch Geometric (PyG) library. Training and testing are performed on a compute node of the High Performance Computing (HPC) cluster Loewenburg, using a single nVidia Tesla V100 GPU.
There are three different surrogate models proposed, and each of them tackles the unpooling limitation in Graph U-Net by using a k-NN approach for upsampling, as used in PointNet++ [35], with k set to 3. The downsampled features at different depths (levels of coarsening) are stored so that the upsampled node co-ordinates required for k-NN interpolation can be obtained from $[c_x, c_y]$ at the downsampled feature of the same depth. The architectures also include skip connections, although they do not perform the task of upsampling here. The three surrogate models differ in the convolutional operations used, as described below.

1. *Graphknn* uses GCNConv layers as used in Graph U-Nets to perform convolutions and executes the upsampling operation with the help of k-NN interpolation.

2. The *SAGEknn* surrogate model replaces the GCNConv layers of Graph U-Net with SAGEConv layers from GraphSAGE [14] and uses k-NN interpolation for unpooling.

3. In *MoNetknn*, we carry out convolutions using the GMMConv layers from the MoNet [29] architecture, along with k-NN interpolation for upsampling.

We perform hyperparameter tuning for each of the architectures to arrive at the ideal choice of design parameters. All the models showcase their best performance for the same hyperparameter configuration - 48 channels, 3 hidden layers each with a pooling ratio of 0.5, ReLU activations and weights initialized by the Kaiming method. We use a batch size of 4 and set up the initial learning rate as 0.0005, to be used along with a Step LR scheduler similar to the baseline surrogates. We use the Adam optimizer and train on the RMSE loss.

4.4. Results and discussion

We perform a 9-fold cross-validation on the training and validation datasets. Figures 4.7 and 4.8 depict the predicted velocity fields and the associated absolute errors obtained from the three surrogates for four different flow conditions. Figure 4.6 provides an overview of the predicted pressure and velocity fields for all the surrogate models discussed here for a case showing Coanda adhesion.

We note that the duration of steady-state simulations is extremely long, about 8.5 hours.

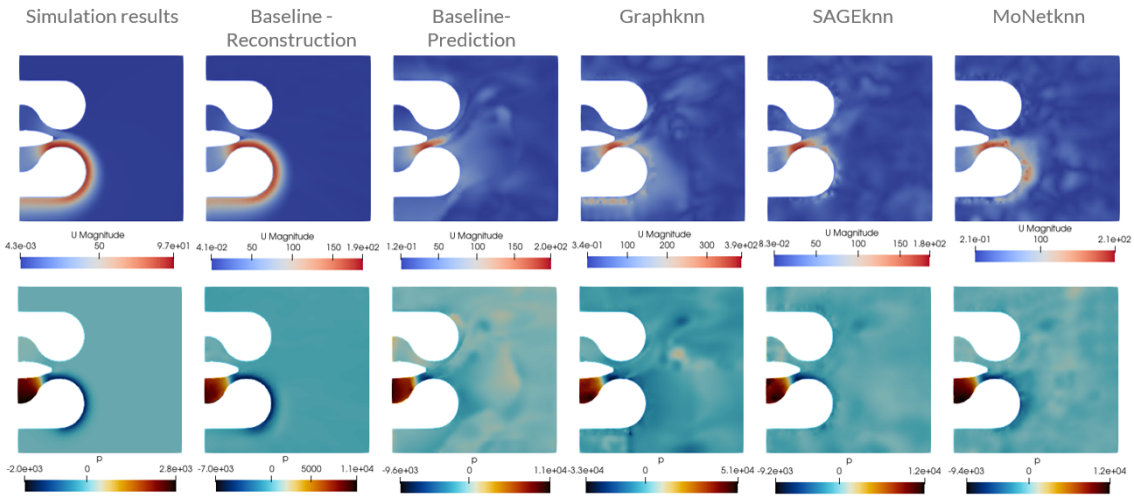


Figure 4.6.: Predicted velocity and pressure fields for the case with (Inlet 1, Inlet 2) = (35 m/s, 5 m/s). The leftmost column represents simulation results, whereas the subsequent columns correspond to the predictions of the respective architectures.

Roughly eight cases are concurrently processed on the Loewenburg HPC cluster with an Intel Xeon Gold 6130F Central Processing Unit (CPU), running at a base frequency of 2.10 GHz. Subsequent simulation cases queued and initiated as prior ones conclude using the Simple Linux Utility for Resource Management (SLURM) scheduler. A simulation takes around 32 minutes on average in the parallel configuration of `OpenFOAM`, amounting to around 8.5 hours for the cumulative simulations of 120 cases.

Investigating further into the test cases, we compute the test loss of each sample in the test dataset and tabulate them in Table 4.6.

Table 4.4.: Evaluation metrics of 1. *Graphknn*, 2. *SAGEknn*, and 3. *MoNetknn* architectures.

Model	**Training Loss**	**Validation Loss**	**Test Loss**
Graphknn	0.134412	0.143149	0.136981
SAGEknn	0.123704	0.138665	0.133959
MoNetknn	0.129066	0.138896	0.146817

Table 4.5.: Time taken for training and testing of (1) *Graphknn*, (2) *SAGEknn*, (3) *MoNetknn* models, (4) Transition, and (5) Steady-state simulations. Note that the training time for (4) and (5) refers to simulation times.

Setup	**Training time (hours)**	**Test time (seconds)**
Graphknn	3.88	0.09157
SAGEknn	3.2	0.09382
MoNetknn	4.15	0.09413
Steady-state	8.5	-
Transition	0.3	-

4.5. Key observations and inference

We make the following key observations in regard to the evaluation metrics obtained:

1. The comparative analysis reveals a positive trend: all three surrogate models exhibit superior performance over the baseline model in predicting steady-state solutions.

2. *SAGEknn* stands out as the most balanced model, offering both the best generalization performance (as indicated by the lowest test loss) and the shortest training time. This reflects an optimal blend of computational efficiency and model precision making it the most effective surrogate model discussed here.

3. Out of the other two models, *MoNetknn* fits the training data better (as indicated by the lower training loss), but has a higher test loss, indicating potential overfitting and poor generalization. *Graphknn* not only takes less time to train but also generalizes better compared to *MoNetknn*, making it the next effective model.

4. Test times for all models are impressively short, indicating that once trained, any of the models can make predictions quickly.

Table 4.6.: Individual loss estimation of test samples in 1. *Graphknn*, 2. *SAGEknn*, 3. *MoNetknn* models and identification of noisy data (highlighted in red). For each test case, the value in bold corresponds to the best test loss achieved out of the three models. The average test loss, excluding noise data, is also computed and the best prediction case for each model are stated.

Case No. (Inlet 1, Inlet 2)	*Graphknn*	*SAGEknn*	*MoNetknn*
10 (18 m/s, 27 m/s)	0.1627682	**0.141760**	0.161019
85 (45 m/s, 20 m/s)	0.227854	**0.294137**	0.256933
96 (45 m/s, 10 m/s)	0.146616	0.153643	0.158489
115 (27 m/s, 3 m/s)	0.111698	**0.100347**	0.116409
93 (32 m/s, 8 m/s)	0.1099330	**0.095786**	0.125213
31 (6 m/s, 24 m/s)	0.115487	**0.073103**	0.108589
94 (36 m/s, 9 m/s)	0.130061	**0.120798**	0.152372
88 (15 m/s, 5 m/s)	**0.123582**	0.133565	0.128406
97 (15 m/s, 3 m/s)	0.134011	**0.122433**	0.144798
110 (35 m/s, 5 m/s)	0.111097	**0.0962448**	0.122681
48 (3 m/s, 21 m/s)	**0.124982**	0.135190	0.140194
80 (20 m/s, 10 m/s)	0.145688	**0.140505**	0.146698
Test Loss	0.136981	**0.133959**	0.146817
Test Loss (w/o noise)	0.128720	**0.119398**	0.136806
Best Case	Case 93	Case 31	Case 31

5. As highlighted in the table, Case 85 has significantly higher error values across all three models compared to the other cases, which is flagged as potential noise or an outlier. This suggests that there's something inherently different or problematic with this particular case, which the models are struggling to predict accurately.

6. The test loss of the models shows a decrease when potentially noisy data is removed, suggesting that outliers like Case 85 are adversely affecting the model's performance. This is especially significant for *SAGEknn*, where the test loss drops from 0.1339 to 0.1193, a substantial reduction.

7. *SAGEknn* not only has the lowest test loss but also shows the greatest improvement when noise is removed, suggesting it has better noise handling or generalization capability compared to the other two models.

8. Case 31 seems to be the overall best scenario for all models, indicating this particular case is well represented in the training data.

We proceed to make further observations based on visual inspection,

1. All the models are able to capture key physical flow features critical to jet deflection

and adhesion to Coanda surfaces such as points of attachment and separation. This indicates that they have learned significant aspects of nozzle flow dynamics.

2. It is noticeable that while qualitative features of the flow are predicted, the quantitative aspects such as exact speed or pressure within the flow field may not be as accurately captured, as indicated by differences in the color intensity between the CFD and model predictions.

3. While comparing the models, *SAGEknn* appears to most closely match the CFD results in terms of capturing the flow direction and adherence to surfaces, which is consistent with the earlier noted superior performance in terms of test loss.

4. The CFD results display sharper gradients near the convergence zone and the Coanda surface, whereas the predictions tend to show smoother gradients.

4.5.1. Checkerboard artifacts

An important observation that is consistent with all three models and flow conditions is the occurrence of a checkerboard pattern in the visualization of velocity fields. We note that these checkerboard patterns are more pronounced in cases with lower velocity ratios compared to those with higher velocity ratios. Among the three models, *SAGEknn* seems to have the least pronounced checkerboard effect, suggesting a better handling of spatial information. *Graphknn*, while slightly exhibiting this pattern, still maintains a good approximation of the flow, balancing the artifact against accurate flow prediction.
The checkerboard pattern observed is usually seen in CNNs due to factors like strided convolutions, which are not inherent to GNNs. It is to be noted that the checkerboard pattern is not observed in the baseline models, which do not implement k-NN interpolation. Hence, we can attribute the source of this discrepancy to k-NN interpolation, which upsamples the graph back to its original fine resolution. This process may introduce regular, grid-like patterns if the interpolation points are evenly spaced or if there is a scale mismatch between the graph's resolution and the visualization grid. Additionally, the value chosen for 'k' in the k-NN algorithm might not sufficiently capture the data's continuity, leading to abrupt changes in interpolated values that manifest as checkerboard artifacts.

In summary, these observations suggest that all models are reasonably well calibrated, with *SAGEknn* slightly outperforming the others. The impact of the outlier Case 85 on the test loss underscores the importance of robust outlier detection and handling in the training process to improve model accuracy and reliability. The checkerboard problem can be mitigated by integrating physics-informed components such as imposing the continuity equation as a constraint in the loss function.
When combined with transitional simulations, properly trained and fine-tuned GNN models can offer a more time-efficient alternative to standalone steady-state simulations. The current suite of steady-state simulations sets a high benchmark in terms of accuracy for

modelling complex physical phenomena. While there is still room for enhancement in accurately capturing the complex physics of turbulent flows, *SAGEknn* emerges as the better alternative as the successor for Graph U-Nets in this scenario.

4.6. Challenges

The endeavor to enhance the performance of Graph U-net architectures through the development of Graphknn, SAGEknn, and MoNetknn models has provided significant insights, albeit alongside challenges reflected in the predictive accuracy. Thus, it remains crucial to reflect on and recognize the factors affecting the models' performance.

- **Training data diversity:** The variety of scenarios and varied parameters in the training data critically influence the model's robustness. Since the model is trained on a limited range of data, it may not generalize well to new, unseen scenarios, leading to poor predictive performance.
- **Boundary conditions:** In the context of simulations for fluid dynamics, accuracy in modelling boundary conditions is essential. Inaccuracies or inadequate modelling may have led to significant errors in predictions around the walls.
- **Data imbalance:** The inlet velocity ratios for the simulations were sampled from the range [1,10] and based on the ratios, the velocities are fixed. By not sampling from a well-defined probability distribution, the model might be missing out on learning the nuances of more complex inlet velocity profiles, which are often encountered in practical applications.
- **Training adequacy:** The extent of training can dictate the model's ability to learn from the data, and is influenced by factors such as dataset size and training duration. For the models, a limitation to only 500 epochs of training and a limited dataset of merely 120 cases might not suffice to converge to a robust solution. This restricted scope of training could hinder the generalization and learning of complex patterns, particularly for complex domains like fluid dynamics.

Extensive datasets with different flow scenarios and longer training periods could offer promising solutions. Future work could also involve employing a more representative dataset sampling strategy, such as sampling from a probability distribution that better captures the variability of real-world conditions. Another suggestion to improve the predictions is to include the pressure of the transition state as a node feature.
Apart from the aforementioned factors, the downsampling operation using top-k pooling may also be a contributing factor to the suboptimal predictions. Feature selection bias, spatial hierarchy distortion, loss of information, and hyperparameter sensitivity are areas of concern regarding this approach. Alternative strategies such as hierarchical pooling,

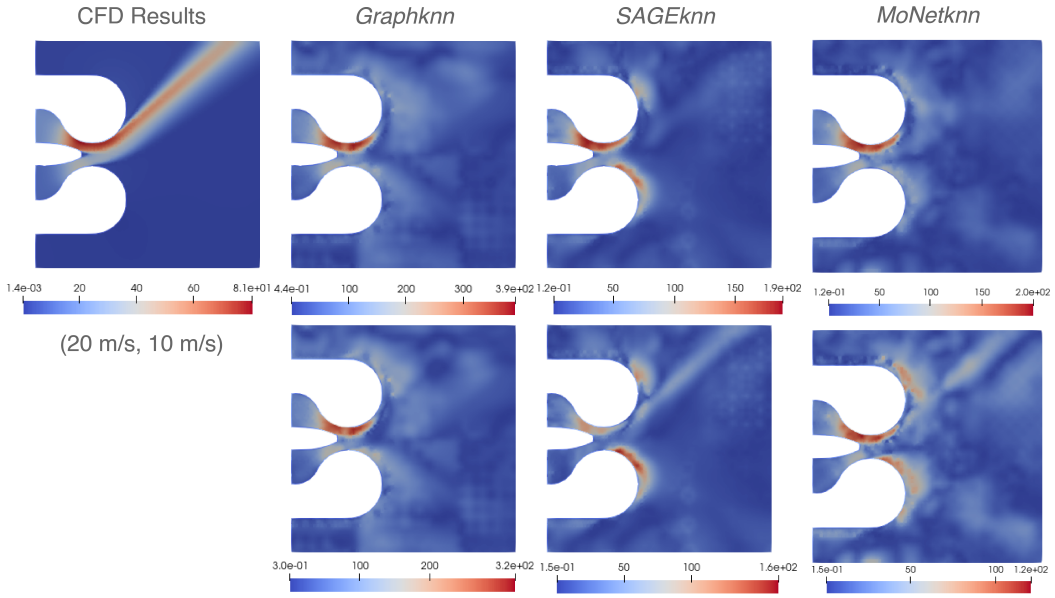
differential pooling or attention mechanisms can be used to tackle the challenges faced by top-k pooling. These considerations serve as a foundation for future research to enhance the performance of GNN architectures in this domain, ensuring that the essential characteristics of the data are captured.

4.7. Hierarchical multi-resolution pooling mechanism in GNN architecture

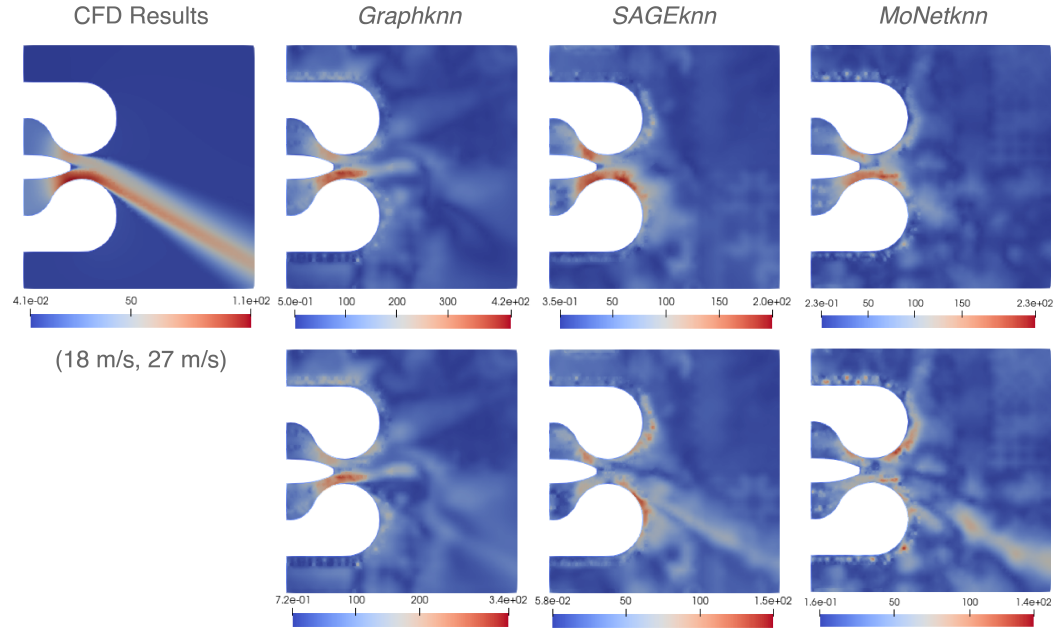
As part of our continuous effort to enhance the performance and applicability of GNNs, we also embarked on an ambitious endeavor aimed at integrating a hierarchical multi-resolution sampling operator. Multi-resolution approaches in the context of GNNs involve operating on graphs at multiple levels of granularity, similar to the multigrid method in numerical analysis. Unlike CNNs, where downsampling operators automatically coarsen the structured mesh, in GNNs, we create a hierarchy of meshes with increasing complexity over the domain of interest. Hence, traditional pooling operators may not be suitable for GNNs, as they focus on selecting nodes to construct a coarse graph, which is unnecessary for mesh data. Instead, we can easily define operators that transform features from one mesh to the next by generating a set of meshes with varying coarseness.
Creating a mesh hierarchy of different levels of coarseness can be performed by well established techniques in numerical analysis. One commonly used algorithm for mesh construction is Delaunay triangulation, which maximizes the minimum angle of all triangles to avoid sliver triangles. This algorithm gradually inserts new nodes into the triangulation and connects them with their neighbors under specific rules. Incremental decimation is another mesh coarsening method that aims to reduce the number of points while preserving specific properties of the original mesh. It iteratively removes one vertex or edge with minimal changes until certain criteria are met.
We then introduce the sampling operator for converting data between two meshes, denoted as M_1 (downsampled mesh) and M_2 (mesh to be downsampled), from the k-NN interpolation proposed in PointNet++ [35]. Liu et al. [26] defines this operator as,

$$\mathbf{f}(\mathbf{z}) = \frac{\sum_{i=1}^{k} w\left(x_i\right) \mathbf{f}\left(x_i\right)}{\sum_{i=1}^{k} w\left(x_i\right)}, \text{ where } w\left(x_i\right) = \frac{1}{\left\|z - x_i\right\|_2} \tag{4.5}$$

However, this aspect of the research is currently in a developmental phase. Preliminary efforts have encountered challenges, particularly in adapting the pooling operation to efficiently process batched data. While the work to fully realize and integrate this new pooling operator within the GNN framework remains underway, the preliminary insights gained and the challenges encountered offer valuable perspectives for advancing GNN methodologies. The ongoing nature of this work underscores the dynamic and evolving landscape of GNN research, where iterative exploration is essential for uncovering new possibilities and overcoming existing limitations.

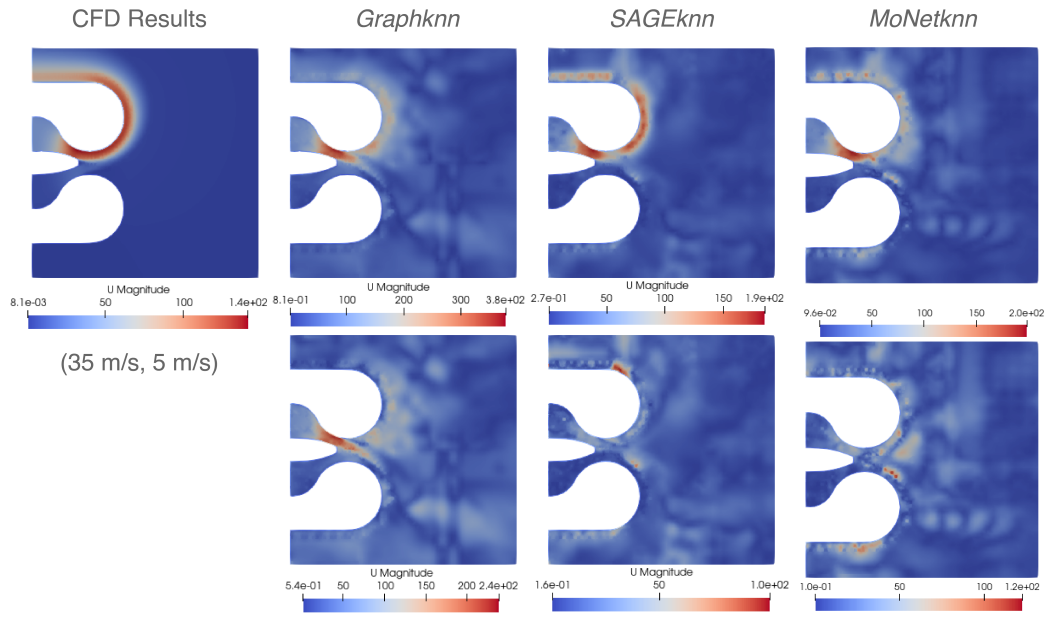


(a) Case (20 m/s, 10 m/s) showing jet deflection to the top Coanda surface.

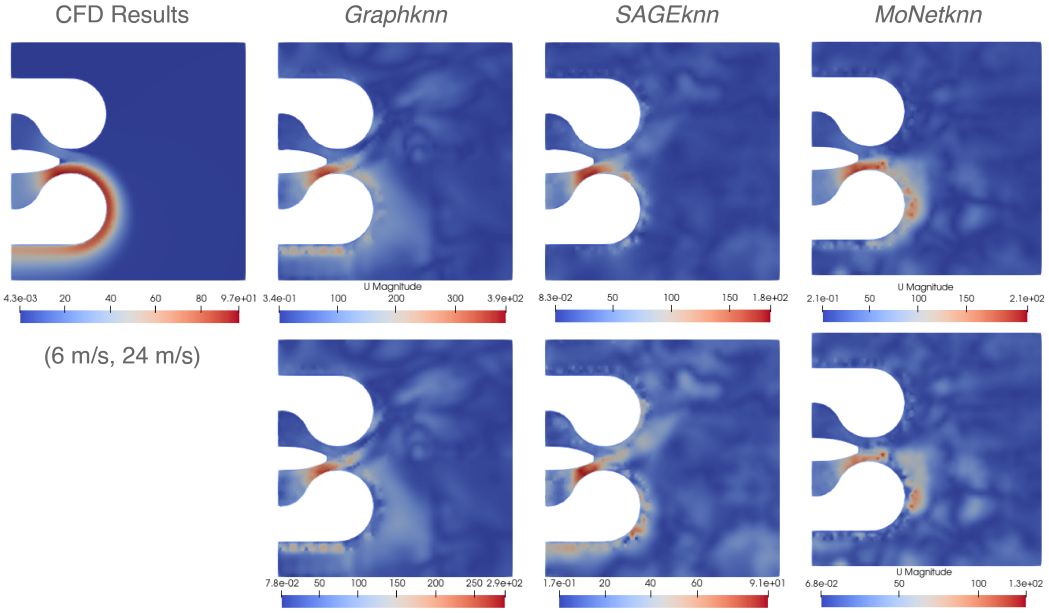


(b) Case (18 m/s, 27 m/s) showing jet deflection to the bottom Coanda surface.

Figure 4.7.: Visualization of velocity fields for two cases with inlet velocities (Inlet 1, Inlet 2): the first row depicts the predictions, and the second row shows the absolute difference between the target and predictions. The top left corner presents the simulation results. The colour bar denotes the magnitude of velocity.



(a) Case (35 m/s, 5 m/s) showing complete adhesion to the top Coanda surface.



(b) Case (6 m/s, 24 m/s) showing complete adhesion to the bottom Coanda surface.

Figure 4.8.: Visualization of velocity fields for two cases with inlet velocities (Inlet 1, Inlet 2): the first row depicts the predictions, and the second row shows the absolute difference between the target and predictions. The top left corner presents the simulation results. The colour bar denotes the magnitude of velocity.

5. Dimensionality reduction and clustering

In this chapter, we explore the application of unsupervised learning techniques, specifically dimensionality reduction and clustering, to analyze and interpret high-dimensional CFD simulation data of nozzle flows. Unsupervised learning, a class of machine learning methods that operate on unlabeled data, aims to discover hidden patterns or intrinsic structures within the data. Our objective is to distill high-dimensional simulation data into insightful, low-dimensional representations and identify distinct patterns through clustering, facilitating a deeper understanding of fluid behavior under various conditions. We use Principal Component Analysis (PCA) [33], and t-Distributed Stochastic Neighbor Embedding (t-SNE) [47] for dimensionality reduction. Subsequently, we apply the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm [8] to the reduced-dimensional data to identify distinct clusters representing various fluid flow behaviors.

5.1. Dimensionality reduction

Dimensionality reduction is crucial in simplifying high-dimensional data, making it amenable to visualization and data analysis. Here, we focus on PCA and t-SNE.

5.1.1. Principal Component Analysis (PCA)

PCA is a linear technique that reduces the dimensions of a dataset by transforming them to a new basis, where the axes are the directions of maximum variance. It effectively compresses the data while attempting to retain the original variance in the data.

5.1.2. t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE, a non-linear technique, converts the high-dimensional Euclidean distances between points into conditional probabilities that represent similarities, aiming to preserve the local structure of the data. The cost function minimized by t-SNE is based on the Kullback-Leibler divergence [6] between the distribution of the high-dimensional data and the distribution of the low-dimensional embedding.

5.2. Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

The DBSCAN algorithm can be employed to identify clusters of varying shapes and sizes based on densities within low-dimensional data, as well as mark outliers in low-density areas. Its idea is to classify points as core points, border points, or outliers, based on the density of their neighborhoods. DBSCAN is defined by a parameter denoting the radius of the neighborhood with respect to a point and another parameter for the minimum number of points required to form a cluster.

5.3. Experiments, results and inference

Dimensionality reduction techniques such as PCA and t-SNE were applied to both transitional and steady-state simulation data constituting the training-validation dataset from surrogate modelling, facilitating the visualization and interpretation of complex flow dynamics. PCA and t-SNE are used to transform the target data to two dimensions, and the results from this transformation can be seen in Figure 5.2 and Figure 5.1. Each data point corresponds to a simulation case and marked by its velocity ratio, which is taken as the ratio of the higher of the two inlet velocities to that of the lower. In the t-SNE plot, we observe clearly distinct clusters of data points on the bottom left and top right corners. The clusters are formed by velocity ratios greater than four, likely indicating the phenomenon of complete adhesion to the Coanda surfaces.
Evidenced by the clustering of points with higher velocity ratios, we propose the hypothesis that as the velocity ratio increases, the tendency of the outflow jet to exhibit complete adhesion to one of the surfaces becomes more pronounced. Simulations with lower velocity ratios appear to be more dispersed across the t-SNE plot, which might indicate a more varied flow behavior at these ratios or a weak adhesion effect. The data does not cluster as tightly as those with higher velocity ratios, implying a case of jet deflection without adhesion to surfaces. We also plot the t-SNE data, categorizing each data point based on its inlet velocities relationship as seen in Figure 5.3. The application of DBSCAN to these low-dimensional representations from the t-SNE transformation provides further evidence for our speculated theories with the identification of distinct flow behavior clusters, based on Coanda adhesion phenomena. An examination of the scatter plot in Figure 5.4 shows two clusters as well as outlier points scarcely distributed between these two regions:

- **Blue cluster (Coanda adhesion to the top surface):** The blue cluster, isolated in the upper quadrant of the plot, likely represents instances of Coanda adhesion to the top surface of the nozzle. The spatial separation of these points from the central mass suggests a specific, consistently observed flow behavior across these simulations.
- **Yellow cluster (Coanda adhesion to the bottom surface):** The cluster circled in yellow, situated in the bottom left of the plot, corresponds to the simulations exhibiting

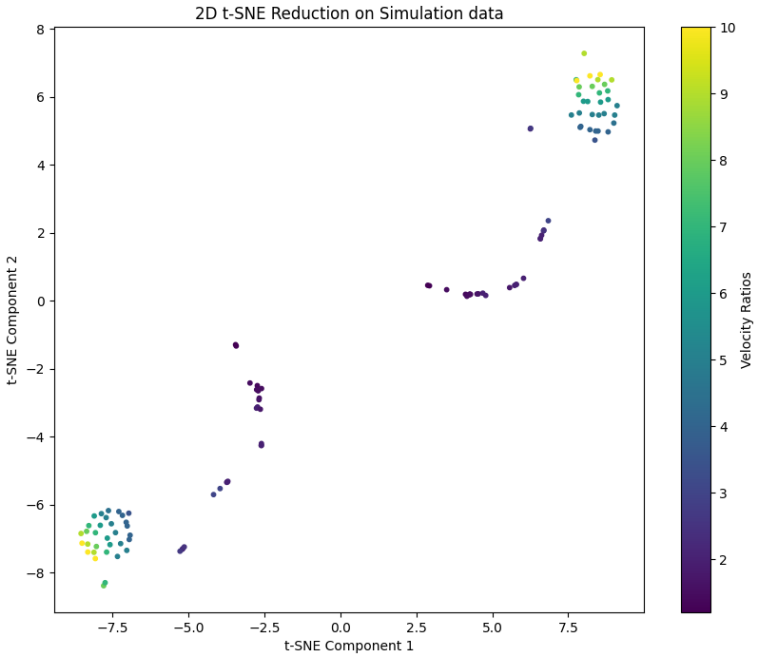


Figure 5.1.: Scatter plot visualizing the 2D t-SNE reduction of steady-state simulation data, categorized by varying velocity ratios. Each point on the plot represents a simulation data sample, and the color coding corresponds to the velocity ratios.

Coanda adhesion to the bottom surface. The compactness and isolated location of this cluster signify a distinct and strong pattern of flow behavior.

- **Purple outlier points (Jet deflection):** The outliers marked in purple, dispersed between the blue and yellow clusters, likely characterize scenarios where the jet deflects towards either surface, representing a transitional behavior that does not culminate in pronounced Coanda adhesion.

We also perform visual inspection of the simulations and classify them into four instances:

1. Complete adhesion to the bottom Coanda surface.
2. Jet deflection and partial adhesion to the bottom Coanda surface.
3. Complete adhesion to the top Coanda surface.
4. Jet deflection and partial adhesion to the top Coanda surface.

It is observed that the cluster on the bottom left corresponds to cases displaying complete adhesion to the bottom Coanda surface whereas the one on the top right are constituted by cases that show complete Coanda adhesion to the top wall. The classification of the

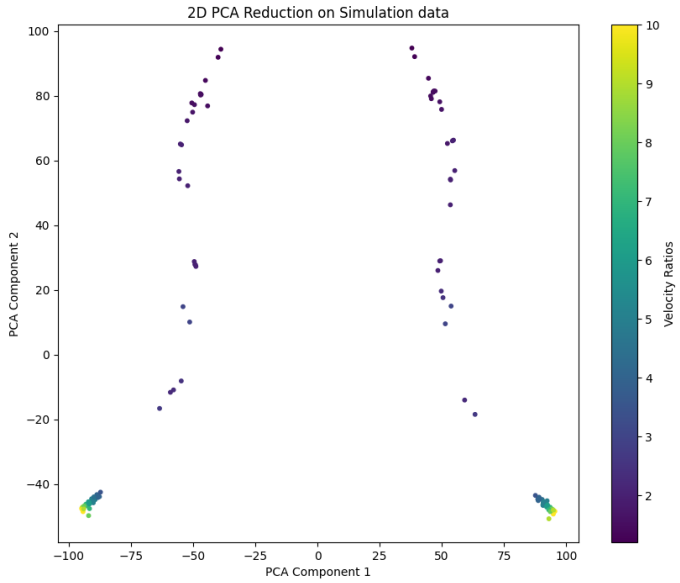


Figure 5.2.: Scatter plot visualizing the 2D PCA reduction of steady-state simulation data, categorized by varying velocity ratios. Each point on the plot represents a simulation data sample, and the color coding corresponds to the velocity ratios.

cases is presented in the Appendix 6.2 serve as the ground truth: cases 89 to 120 show Coanda adhesion to the top wall, whereas cases 29 to 60 display complete adhesion to the bottom surface. We annotate all the points on the t-SNE plot to analyze and evaluate our argument. As seen in Figure 5.5, we observe an exact match in the cluster behavior and associated cases demonstrating a 100% prediction accuracy in clustering the low-dimensional, steady-state data. Thus, we can assert that simulation cases with higher velocity ratios (around four and above) show complete Coanda adhesion to either of the surfaces. Higher Inlet 1 velocity shows adhesion to the top surface and vice-versa.
The effective application of dimensionality reduction and clustering methods, such as t-SNE and DBSCAN, allows for the automation of the identification and classification of instances where Coanda adhesion takes place. This approach is particularly valuable for large datasets where manual classification would be impractical and time-consuming, demonstrating how these techniques can streamline and enhance data analysis processes.
The PCA-reduced data also shows dense regions formed by cases with higher velocity ratios. However, clustering of simply PCA-reduced target data fails to recognize these clusters, as demonstrated by Figure 5.6. This may be attributed to the fact that PCA is a linear dimensionality reduction technique and may sometimes not work effectively for certain datasets when the structure of the data is non-linear. PCA preserves global structure and may not unfold the data in a way that emphasizes the separation between clusters, particularly if the clusters are non-linearly separable in the original high-dimensional space.

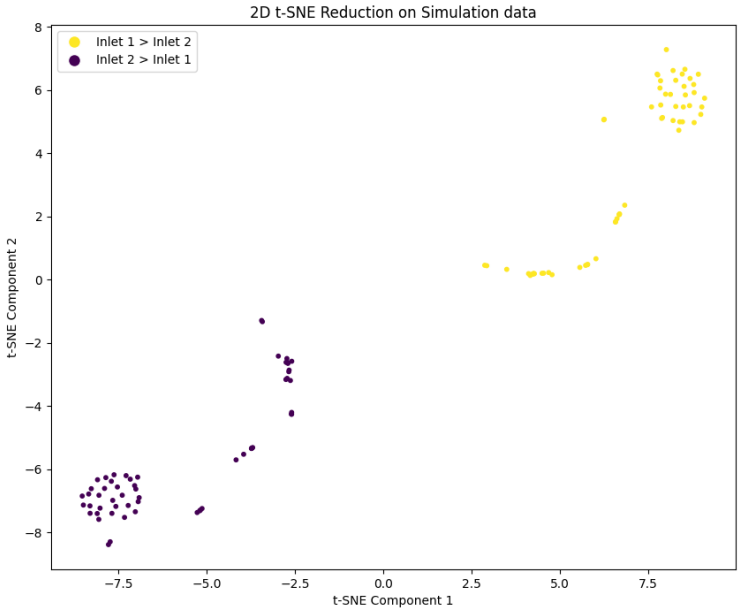


Figure 5.3.: Scatter plot visualizing the 2D t-SNE reduction of steady-state simulation data, categorized by Inlet 1 and Inlet 2 relationship.

t-SNE, on the other hand, is specifically designed to preserve local neighborhood structure and can reveal clusters even in complex datasets with non-linear structures. This result led us to perform a PCA reduction to 50 components, followed by a t-SNE reduction to 2 dimensions. This combination was able to replicate similar results to t-SNE-reduced data, as seen in Figure 5.7. The outcome of clustering from DBSCAN is shown in Figure 5.8.

Additionally, we reduce the simulation data to a one-dimensional representation, as demonstrated in Figure 5.9. We observe that the data points are distinctly clustered even along the single t-SNE component axis, indicating that the t-SNE algorithm has successfully captured and represented the variance within the high-dimensional dataset in a one-dimensional space. We can also see the emergence of distinct clusters, each cluster denoting a type of fluid behavior.

It is to be noted that t-SNE has an element of randomness in the way it projects high-dimensional data into a lower-dimensional space. This randomness is due to the stochastic nature of the algorithm, particularly in the initial placement of points in the low-dimensional space. The orientation and the exact position of the clusters in the map are not fixed and can differ by a rotation or reflection. This is because t-SNE does not preserve the orientation or the exact distances; instead, preserves the local structure of the data and the global relationships between clusters.

For the transitional state, t-SNE did not reveal any distinct clusters, indicating that at this early stage, the flow behaviors do not exhibit clear patterns that can be distinguished.

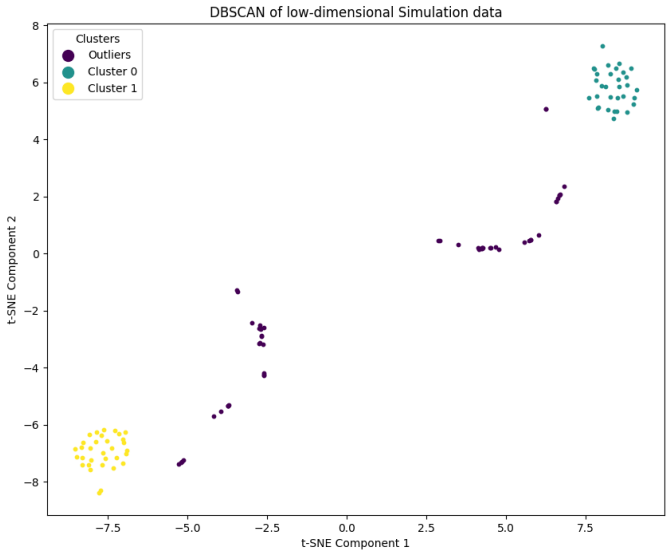


Figure 5.4.: DBSCAN clustering visualized on t-SNE-reduced steady-state simulation data.

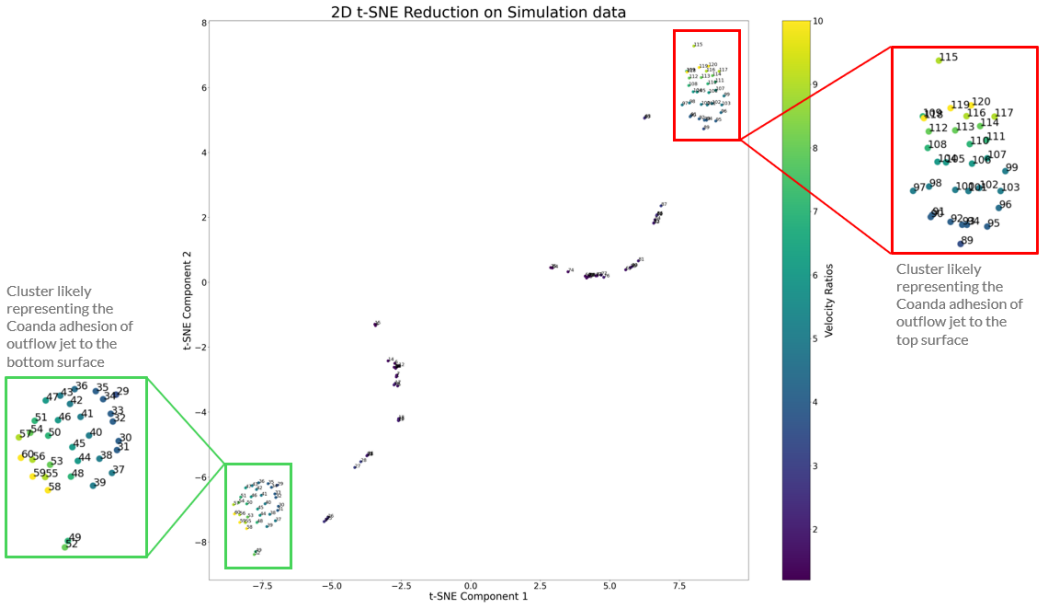


Figure 5.5.: Annotation of each sample point based on its simulation case. Here, the green box captures the cluster that is representative of cases exhibiting complete Coanda adhesion to the top surface, whereas the red box likely shows the cluster comprising the cases marked by complete Coanda adhesion to the bottom wall. Note that the original clusters are zoomed in for readability.

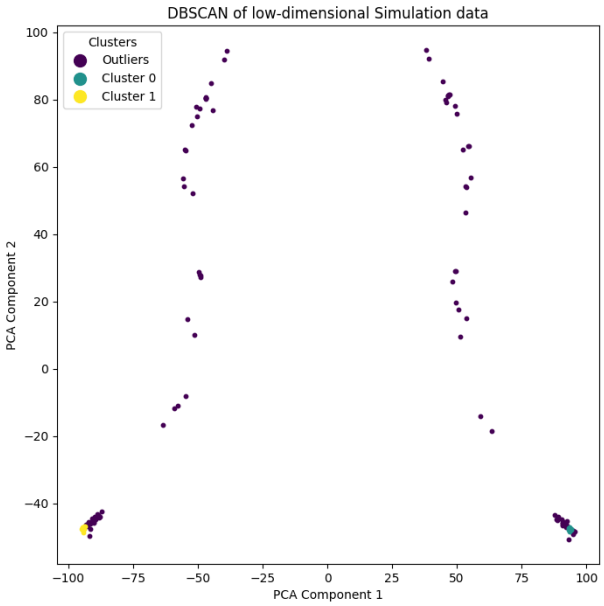


Figure 5.6.: DBSCAN clustering visualized on PCA-reduced steady-state simulation data.

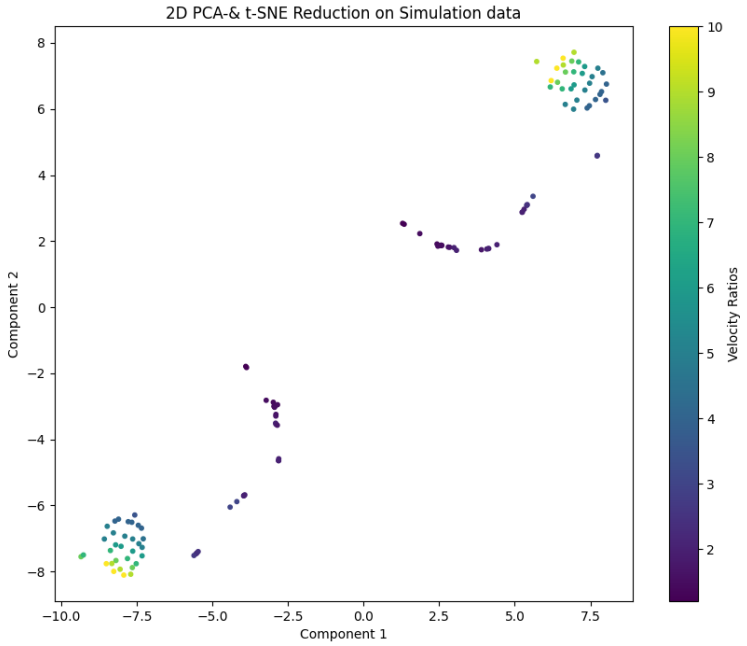


Figure 5.7.: Scatter plot visualizing the subsequent PCA and t-SNE reduction of steady-state simulation data, categorized by varying velocity ratios.

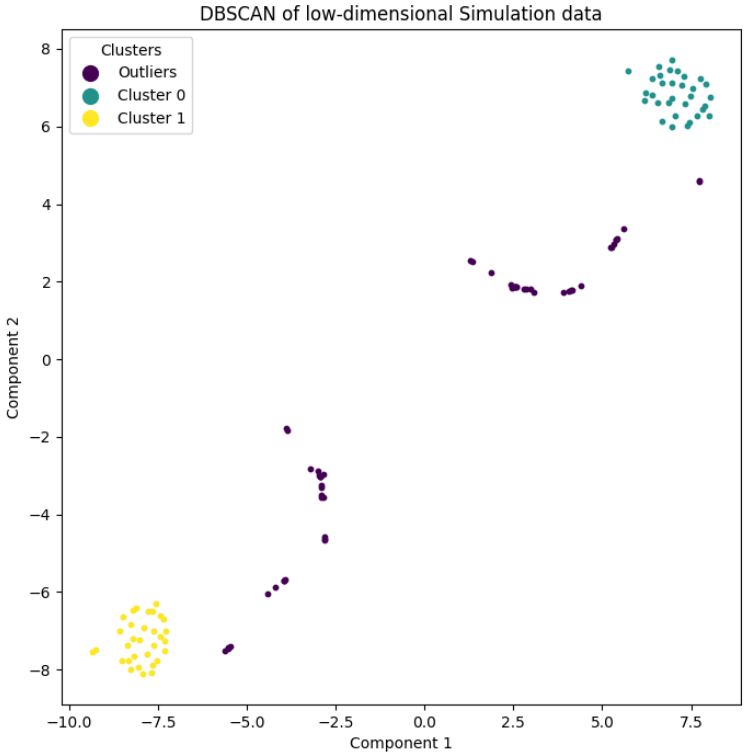


Figure 5.8.: DBSCAN clustering visualized on subsequent PCA and t-SNE reduced steady-state simulation data.

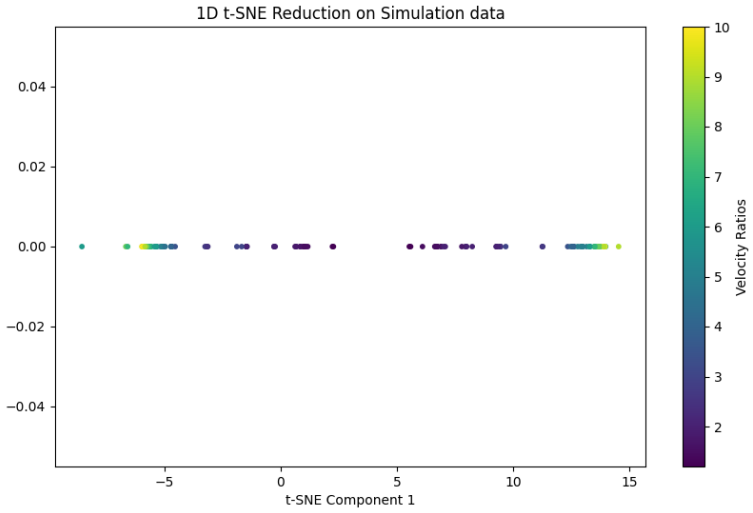


Figure 5.9.: Visualization of 1D t-SNE-reduced steady-state simulation data

Part IV.

Conclusion

6. Conclusion

This thesis undertakes a critical exploration into the application of Graph Neural Networks (GNNs) within turbulence modeling, motivated by the simulation of nozzle flow dynamics on unstructured meshes. Through a series of methodical experiments and analyses, the study attempts to gauge the efficacy of GNNs as surrogate models. In this final chapter, we conclude by reviewing the initial objectives of the research described in Chapter 1 and presenting a summary of our key contributions. Finally, we recommend a roadmap for future work that can address the challenges faced and pave the way for subsequent research efforts.

6.1. Recap of objectives and contributions

At the outset of this thesis, we outlined several key objectives aimed at exploring the integration of GNNs for the application of nozzle flow simulations. As we conclude, it is important to revisit these objectives to assess the contributions of this research in addressing the initial goals:

1. **Develop a GNN model for predicting nozzle flow simulation quantities:**
   Contribution:
   - Computational modelling and simulation of the HOMER nozzle and generation of a CFD dataset for nozzle flow simulations comprising unstructured mesh data for various flow conditions.
   - Extraction of CFD simulation data and transformation of the raw data from unstructured meshes into graph-structured data.
   - Successful development and implementation of various GNN models, including a baseline Graph U-Net and three other architectures — Graphknn, SAGEknn, and MoNetknn. These models were trained to predict key simulation quantities, demonstrating GNNs' potential in modelling fluid dynamics.
2. **Investigate the accuracy, efficiency and feasibility of the surrogate models:**
   Contribution:
   - Transformation of graph data back to CFD format for visual analysis of predictions with respect to targets, facilitating a clear visual framework to assess the accuracy of the surrogate models.

- Performed comparative analysis across different GNN architectures to evaluate their performance in terms of both training and testing losses, which serves as a measure of accuracy, as well as runtimes, which elucidate the models' efficiency. This also highlights their potential to significantly reduce the computational time and resources required for fluid dynamics simulations.

3. **Perform clustering on low-dimensional data to classify simulations:**
   Contribution:
   - Applied dimensionality reduction using PCA and t-SNE, enabling the visualization of fluid dynamics patterns from high-dimensional data as well as categorization of the simulations based on velocity ratios.
   - Applied the DBSCAN clustering technique to low-dimensional data to identify distinct fluid flow behavior clusters. We achieved 100% prediction accuracy in clustering steady-state data, accurately distinguishing between complete adhesion to top or bottom Coanda surfaces and the outliers depicting partial adhesion. This highlights the precision of the approach in automating the classification of flow dynamics.

4. **Investigation of advanced GNN architectures for enhanced model performance:**
   Contribution:
   - Explored the optimization of GNN architectures using a sampling operator for hierarchical multi-resolution feature learning.

The results obtained from these experiments underscore the nascent potential of GNNs within the realm of CFD, albeit with the acknowledgment that there remains much room for improvement. The process of model evaluation and the insights gained from these comparisons serve as a modest contribution to the broader effort to integrate machine learning techniques with CFD simulations. By systematically addressing each objective, we offer valuable insights into the potential and challenges of GNN applications in fluid dynamics. Through this endeavor, we contribute to the ongoing discourse in the intersection of deep learning and CFD, setting a foundation for future investigations to build upon in pursuit of more efficient, accurate, and broadly applicable models.

6.2. Future directions

The research presented in this thesis asserts the transformative potential of GNNs as surrogate models in the domain of CFD. Looking forward, it is imperative to expand upon this work by exploring more complex geometries and further refining GNN architectures to enhance their generalizability and predictive capabilities across a broader spectrum of fluid dynamics applications. Several paths for future work emerge, each aimed at advancing our understanding of GNNs in fluid dynamics:

1. **Enhancement of GNN architecture for improved accuracy:** One of the primary objectives moving forward is the development of improved GNN architectures that enhances prediction accuracy. This involves exploring novel graph convolutional layers, attention mechanisms, and network structures that can more effectively capture the complexities of fluid dynamics.

2. **Prediction of extended flow quantities of interest:** An additional critical objective for future work is the extension of the GNN model's capabilities to predict a broader array of flow quantities. Specifically, we can tailor the model to accurately forecast values such as turbulent viscosity ν_t,turbulent kinetic energy k, and the specific rate of dissipation ω. These quantities are fundamental in the analysis and modelling of turbulent flows, providing deeper insights into the behavior of fluids in various engineering applications.

3. **Optimization of model training times:** Another crucial area of research is the optimization of the GNN model to reduce training times without compromising accuracy. We can leverage parallel computing and advanced hardware accelerators to further enhance training efficiency, enabling the model to learn from larger datasets in shorter time frames.

4. **Development of physics-informed GNNs:** Incorporating the governing equations of fluid dynamics into the model's loss function presents a promising approach to enhance prediction accuracy. By developing a physics-informed GNN, the model can leverage both data-driven learning and the inherent physics of fluid flow, ensuring more accurate and physically plausible predictions. This approach aids in better capturing complex phenomena associated with turbulent flows and also in improving the model's interpretability and reliability.

5. **Model generalization and adaptability:** The key direction for future research is to enhance the generalization and adaptability of GNN models to different nozzle configurations, flow conditions, and novel scenarios. Incorporating transfer learning by leveraging pre-trained models on extensive datasets could aid in generalization across various nozzle flow scenarios. Ultimately, the aim is to develop robust models capable of accurately simulating fluid behavior across a wide range of scenarios with minimal need for retraining.

The future directions outlined above offer a roadmap for further investigation and reflect the iterative nature of research in integrating deep learning with fluid dynamics. In summary, this thesis not only fulfills its objectives but also opens new avenues for research, underscoring the pivotal role of DL in advancing CFD towards more efficient and accurate simulations. The technical achievements documented herein confirm the feasibility of GNNs as powerful surrogate models, signifying a major step forward in our ability to simulate and understand complex fluid dynamics phenomena.

Appendix

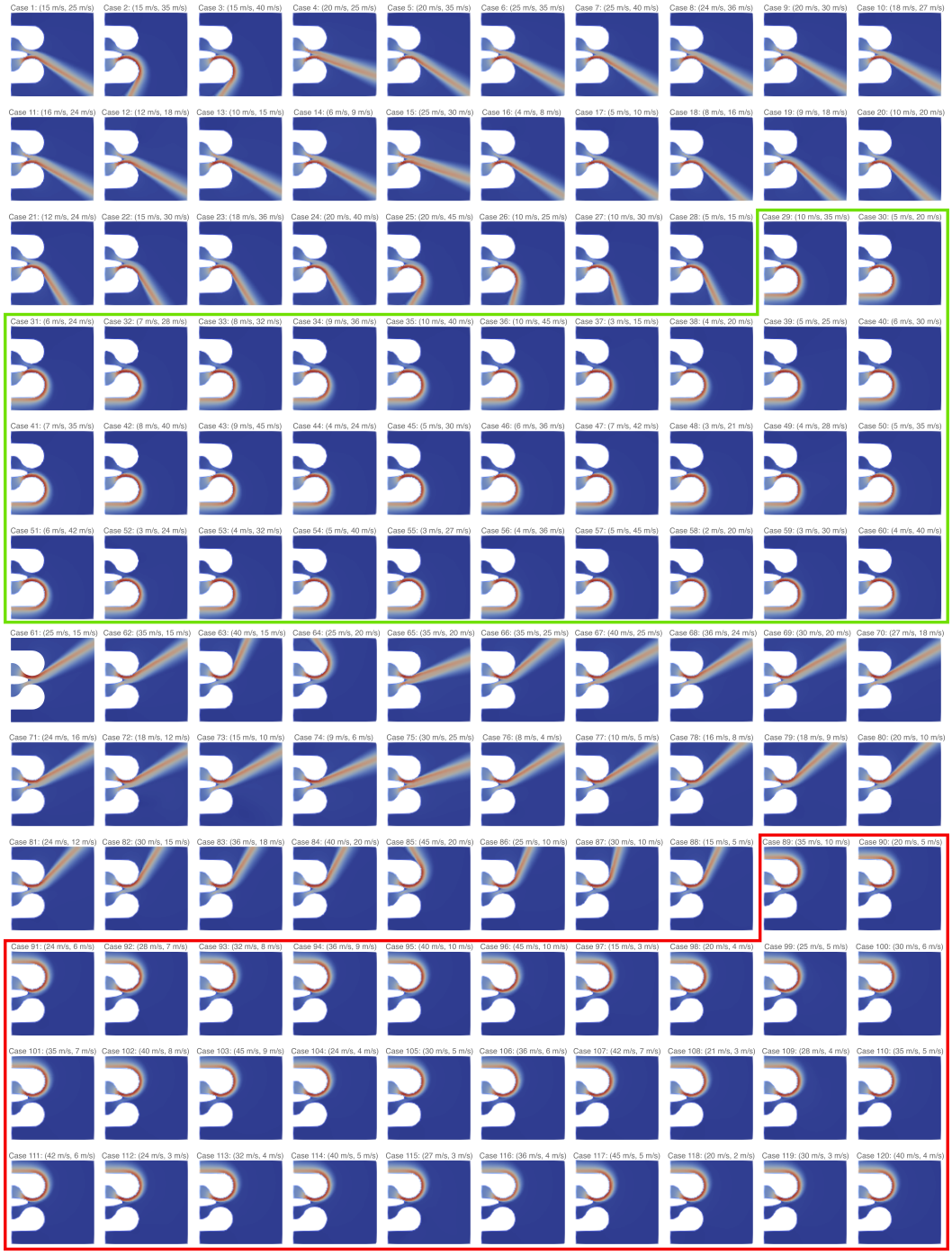


Figure 1.: Categorization of 120 simulation cases - the simulation cases encapsulated by a green box represent complete Coanda adhesion to the bottom surface, and those enclosed in a red box represents complete adhesion to the top wall.

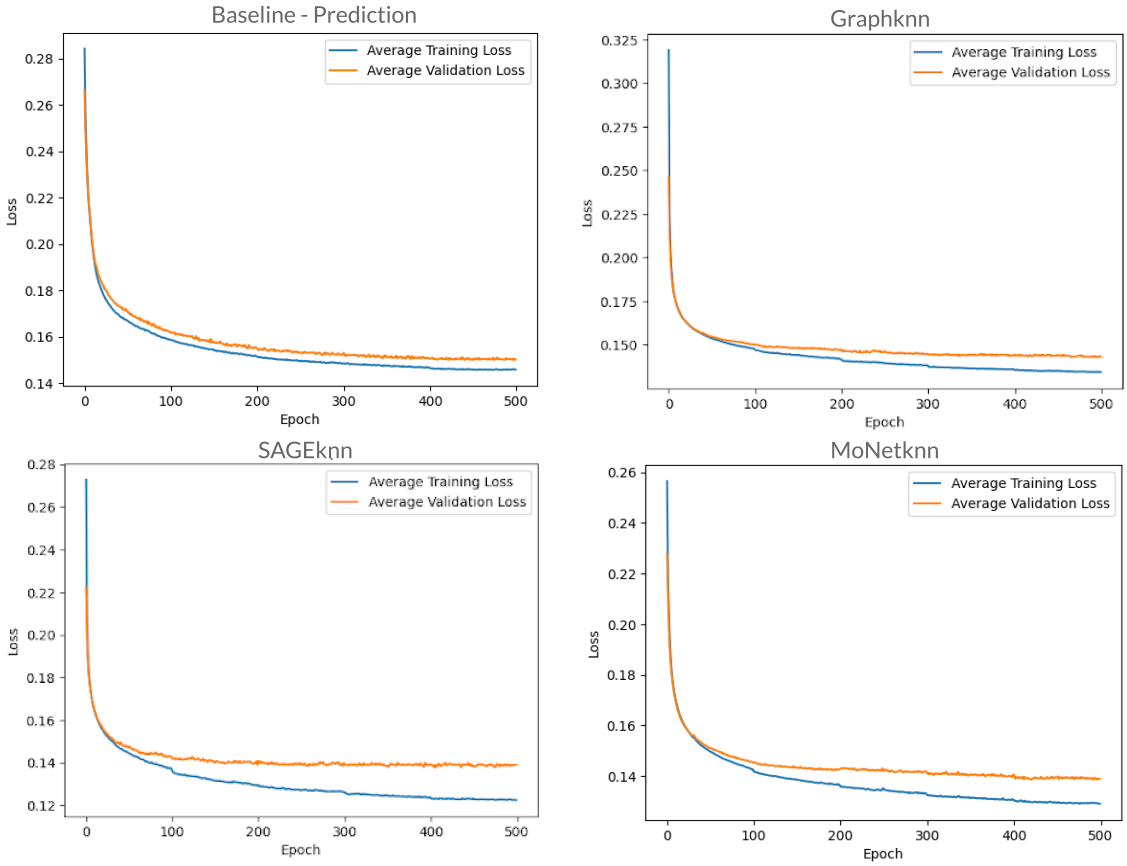


Figure 2.: Loss curves showing the average training and validation loss characteristics over 9 folds across 500 epochs of the prediction models: (a) Baseline (b) *Graphknn* (c) *SAGEknn* (d) *MoNetknn*

List of Acronyms

AdaGrad Adaptive Gradient. 24

Adam Adaptive Moment Estimation. 24

ANN Artificial Neural Networks. 18

CFD Computational Fluid Dynamics. 3

CNN Convolutional Neural Networks. 4

COO Co-Ordinate list. 36

CPU Central Processing Unit. 45

DBSCAN Density-Based Spatial Clustering of Applications with Noise. 53

DL Deep Learning. 3

DNS Direct Numerical Simulation. 3

FDM Finite Difference Method. 13

FEM Finite Element Method. 13

FVM Finite Volume Method. 13

GCNN Graph Convolutional Neural Networks. 5

GNN Graph Neural Networks. 3

HOMER High-speed Orienting Momentum with Enhanced Reversibility. 5

HPC High Performance Computing. 44

k-NN k - Nearest Neighbors. 32, 44

LES Large Eddy Simulation. 4

LOOCV Leave One Out Cross-Validation. 22

MAE Mean Absolute Error. 25

ML Machine Learning. 3

MLP Multi Layer Perceptrons. 19

MSE Mean Squared Error. 25

NN Neural Networks. 18

NSE Navier-Stokes Equations. 3

PCA Principal Component Analysis. 53

PDE Partial Differential Equations. 3

PINN Physics-Informed Neural Networks. 4

RANS Reynolds Averaged Navier-Stokes. 12

ReLU Rectified Linear Unit. 18

RMSE Root Mean Squared Error. 25

RMSProp Root Mean Squared Propagation. 24

RNN Recurrent Neural Networks. 4

ROM Reduced Order Modelling. 3

SGD Stochastic Gradient Descent. 24

SIMPLE Semi-Implicit Method for Pressure Linked Equations. 14

SLURM Simple Linux Utility for Resource Management. 45

SST Shear Stress Transport. 13

t-SNE t-Distributed Stochastic Neighbor Embedding. 53

List of Symbols

A Adjacency matrix. 29

$\mathbf{B}^{(l)}$ Learnable bias matrix of layer l in graph convolutional operation. 31

$c_{x,i}$ X co-ordinate of node i. 37

$c_{y,i}$ Y co-ordinate of node i. 37

D Number of features per node. 37

E Set of edges in a Graph Neural Network. 28

ϵ Rate of dissipation of turbulent kinetic energy. 12

η Learning rate. 24

F_X Number of input channels. 37

F_Y Number of output channels. 37

G Graph in a Graph Neural Network. 28

$h_i^{(l)}$ Feature vector of node i at layer l. 31

k Turbulent kinetic energy. 12

δ_{ij} Kronecker delta function. 12

$\mathcal{L}(\theta)$ Loss function. 23

λ Regularization strength. 21

μ Dynamic viscosity. 11

N Number of nodes in a graph. 36

$\mathcal{N}(i)$ Set of neighboring nodes of node i. 31

$\gamma_{\text{tag},i}$ Type of node i. 37

ν Kinematic viscosity. 11

ν_t Turbulent viscosity. 12

ω Specific rate of turbulence dissipation. 13

p Pressure. 11

ρ Density of the fluid. 11

σ Activation function. 18

θ Parameters of the neural network being optimized. 23

u_i Flow velocity in the spatial direction x_i. 11

V Set of nodes in a Graph Neural Network. 28

W Edge weight matrix. 29

$\mathbf{W}^{(l)}$ Learnable weight matrix of layer l in graph convolutional operation. 31

X Node feature matrix. 29

y_i Ground truth. 23

$\hat{y}_i$ Prediction of the neural network. 23

Bibliography

[1] Peter Benner, Serkan Gugercin, and Karen Willcox. A survey of projection-based model reduction methods for parametric dynamical systems. *SIAM Review*, 57(4):483–531, 2015.

[2] Gal Berkooz, Philip Holmes, and John L. Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual Review of Fluid Mechanics*, 25(1):539–575, 1993.

[3] Léon Bottou. Large scale online learning. *Advances in neural information processing systems*, 22:217–224, 2010.

[4] Michael P. Brenner, Jeff D. Eldredge, and Jonathan B. Freund. Perspective on machine learning for advancing fluid mechanics. *Physical Review Fluids*, 4(10):100501, 2019.

[5] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52(1):477–508, 2020.

[6] Imre Csiszár. I-divergence geometry of probability distributions and minimization problems. *The annals of probability*, pages 146–158, 1975.

[7] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. In *Journal of Machine Learning Research*, volume 12, pages 2121–2159, 2011.

[8] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[9] Charles L. Fefferman. Existence and smoothness of the navier-stokes equation. *The Clay Mathematics Institute*, 2006. Millennium Prize Problems.

[10] Joel H. Ferziger and Milovan Perić. *Computational Methods for Fluid Dynamics*. Springer, 2002.

[11] FreeCAD Community. Freecad 0.19: An open source parametric 3d cad modeler, 2021. Accessed: 22.12.2023.

[12] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, 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, 13–15 May 2010. PMLR.

[13] Luca Guastoni, Alejandro Güemes, Andrea Ianiro, Stefano Discetti, Philipp Schlatter, Hossein Azizpour, and Ricardo Vinuesa. Convolutional-network models to predict wall-bounded turbulence from wall quantities. *Journal of Fluid Mechanics*, 928, 2021.

[14] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

[15] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2 edition, 2009. Discussion on k-fold cross-validation.

[16] Juncai He and Jinchao Xu. Mgnet: A unified framework of multigrid and convolutional neural network. *Science China Mathematics*, 62(7):1331–1354, May 2019.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[19] Emre Kara and Hüdai Erpulat. Experimental investigation and numerical verification of coanda effect on curved surfaces using co-flow thrust vectoring. *International Advanced Researches and Engineering Journal*, 5:72–78, 2021.

[20] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[21] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[22] J. Nathan Kutz. Deep learning in fluid dynamics. *Journal of Fluid Mechanics*, 814:1–4, 2017.

[23] B. E. Launder and D. B. Spalding. The numerical computation of turbulent flows. *Computer Methods in Applied Mechanics and Engineering*, 3(2):269–289, 1974.

[24] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[25] Julia Ling, Andrew Kurzawski, and Jeremy Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166, 2016.

[26] Wenzhuo Liu, Mouadh Yagoubi, and Marc Schoenauer. Meta-learning for airflow simulations with graph neural networks. *arXiv preprint arXiv:2306.10624*, 2023.

[27] Michele Milano and Petros Koumoutsakos. Neural network modeling for near wall turbulent flow. *Journal of Computational Physics*, 182(1):1–26, 2002.

[28] Parviz Moin and Krishnan Mahesh. *Direct Numerical Simulation: A Tool in Turbulence Research*. Annual Review of Fluid Mechanics, Annual Reviews, 1998.

[29] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. *CoRR*, abs/1611.08402, 2016.

[30] BG Newman. The deflection of plane jets by adjacent boundaries-coanda effect. *Boundary layer and flow control*, 1961.

[31] Francis Ogoke, Kazem Meidani, Amirreza Hashemi, and Amir Barati Farimani. Graph convolutional networks applied to unstructured flow field data. *Machine Learning: Science and Technology*, 2(4):045020, sep 2021.

[32] Eric J. Parish and Karthik Duraisamy. A paradigm for data-driven predictive modeling using field inversion and machine learning. *Journal of Computational Physics*, 305:758–774, 2016.

[33] Karl Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(11):559–572, 1901.

[34] Stephen B. Pope. *Turbulent Flows*. Cambridge University Press, 2000.

[35] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *CoRR*, abs/1706.02413, 2017.

[36] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[37] O. Reynolds. On the dynamical theory of incompressible viscous fluids and the determination of the criterion. *Philosophical Transactions of the Royal Society of London. A*, 186:123–164, 1895.

[38] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[39] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. MIT Press, 1986.

[40] Peter J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28, 2010.

[41] Joseph Smagorinsky. General circulation experiments with the primitive equations: I. the basic experiment. *Monthly Weather Review*, 91(3):99–164, 1963.

[42] Joe F. Thompson, Bharat K. Soni, and Nigel P. Weatherill. Automatic and adaptive mesh generation for curved domains. *Applied Mechanics Reviews*, 51(11):669–704, 1998.

[43] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 2012. Available online: https://www.coursera.org/course/neuralnets.

[44] Brendan D. Tracey, Karthikeyan Duraisamy, and Juan J. Alonso. *A Machine Learning Strategy to Assist Turbulence Model Development*. 2013.

[45] Michele Trancossi and Antonio Dumas. A.c.h.e.o.n.: Aerial coanda high efficiency orienting-jet nozzle. *SAE Technical Papers*, 10 2011.

[46] Nathaniel Trask, Ravi G. Patel, Ben J. Gross, and Paul J. Atzberger. Gmls-nets: A framework for learning from unstructured data. *CoRR*, abs/1909.05371, 2019.

[47] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[48] Ze Jia Zhang and Karthikeyan Duraisamy. *Machine Learning Methods for Data-Driven Turbulence Modeling*.