Graph Neural Networks for fluid mechanics: data-assimilation and optimization

Department of Mechanics, Mathematics and Management
MECHANICAL AND MANAGEMENT ENGINEERING
Ph.D. Program
SSD: ING-IND/06–FLUID DYNAMICS

Final Dissertation

Graph Neural Networks for fluid mechanics: data-assimilation and optimization

---

by
Michele Quattromini:

Supervisors:

Prof. Stefania Cherubini

Prof. Caroline Nore

Coordinator of Ph.D. Program:
Prof. Giuseppe Casalino

Course n°37, 01/11/2021-31/10/2024

Politecnico di Bari

LIBERATORIA PER L'ARCHIVIAZIONE DELLA TESI DI DOTTORATO

Al Magnifico Rettore
del Politecnico di Bari

Il sottoscritto **Michele Quattromini** nato a **Altamura** il **30/06/1992**
residente a **Altamura** in via **fratelli Scarati n.5** e-mail **michele.quattromini@poliba.it**
iscritto al 3° anno di Corso di Dottorato di Ricerca in **Ingegneria Meccanica** ciclo **XXXVII**
ed essendo stato ammesso a sostenere l'esame finale con la prevista discussione della tesi dal titolo:

Graph Neural Networks for fluid mechanics: data-assimilation and optimization

DICHIARA

1) di essere consapevole che, ai sensi del D.P.R. n. 445 del 28.12.2000, le dichiarazioni mendaci, la falsità negli atti e l'uso di atti falsi sono puniti ai sensi del codice penale e delle Leggi speciali in materia, e che nel caso ricorressero dette ipotesi, decade fin dall'inizio e senza necessità di nessuna formalità dai benefici conseguenti al provvedimento emanato sulla base di tali dichiarazioni;
2) di essere iscritto al Corso di Dottorato di ricerca **Ingegneria Meccanica** ciclo **XXXVII**, corso attivato ai sensi del "*Regolamento dei Corsi di Dottorato di ricerca del Politecnico di Bari"*, emanato con D.R. n.286 del 01.07.2013;
3) di essere pienamente a conoscenza delle disposizioni contenute nel predetto Regolamento in merito alla procedura di deposito, pubblicazione e autoarchiviazione della tesi di dottorato nell'Archivio Istituzionale ad accesso aperto alla letteratura scientifica;
4) di essere consapevole che attraverso l'autoarchiviazione delle tesi nell'Archivio Istituzionale ad accesso aperto alla letteratura scientifica del Politecnico di Bari (IRIS-POLIBA), l'Ateneo archivierà e renderà consultabile in rete (nel rispetto della Policy di Ateneo di cui al D.R. 642 del 13.11.2015) il testo completo della tesi di dottorato, fatta salva la possibilità di sottoscrizione di apposite licenze per le relative condizioni di utilizzo (di cui al sito http://www.creativecommons.it/Licenze), e fatte salve, altresì, le eventuali esigenze di "embargo", legate a strette considerazioni sulla tutelabilità e sfruttamento industriale/commerciale dei contenuti della tesi, da rappresentarsi mediante compilazione e sottoscrizione del modulo in calce (Richiesta di embargo);
5) che la tesi da depositare in IRIS-POLIBA, in formato digitale (PDF/A) sarà del tutto identica a quelle **consegnate** ai componenti della commissione per l'esame finale e a qualsiasi altra copia depositata presso gli Uffici del Politecnico di Bari in forma cartacea o digitale, ovvero a quella da discutere in sede di esame finale, a quella da depositare, a cura dell'Ateneo, presso le Biblioteche Nazionali Centrali di Roma e Firenze e presso tutti gli Uffici competenti per legge al momento del deposito stesso, e che di conseguenza va esclusa qualsiasi responsabilità del Politecnico di Bari per quanto riguarda eventuali errori, imprecisioni o omissioni nei contenuti della tesi;
6) che il contenuto e l'organizzazione della tesi è opera originale realizzata dal sottoscritto e non compromette in alcun modo i diritti di terzi, ivi compresi quelli relativi alla sicurezza dei dati personali; che pertanto il Politecnico di Bari ed i suoi funzionari sono in ogni caso esenti da responsabilità di qualsivoglia natura: civile, amministrativa e penale e saranno dal sottoscritto tenuti indenni da qualsiasi richiesta o rivendicazione da parte di terzi;
7) che il contenuto della tesi non infrange in alcun modo il diritto d'Autore né gli obblighi connessi alla salvaguardia di diritti morali od economici di altri autori o di altri aventi diritto, sia per testi, immagini, foto, tabelle, o altre parti di cui la tesi è composta.

Luogo e data Bari, 08/01/2025 Firma _____

Il sottoscritto, con l'autoarchiviazione della propria tesi di dottorato nell'Archivio Istituzionale ad accesso aperto del Politecnico di Bari (POLIBA-IRIS), pur mantenendo su di essa tutti i diritti d'autore, morali ed economici, ai sensi della normativa vigente (Legge 633/1941 e ss.mm.ii.),

CONCEDE

- al Politecnico di Bari il permesso di trasferire l'opera su qualsiasi supporto e di convertirla in qualsiasi formato al fine di una corretta conservazione nel tempo. Il Politecnico di Bari garantisce che non verrà effettuata alcuna modifica al contenuto e alla struttura dell'opera.
- al Politecnico di Bari la possibilità di riprodurre l'opera in più di una copia per fini di sicurezza, back-up e conservazione.

Luogo e data Bari, 08/01/2025 Firma _____

Firma Relatore _____

Department of Mechanics, Mathematics and Management
MECHANICAL AND MANAGEMENT ENGINEERING
Ph.D. Program
SSD: ING-IND/06–FLUID DYNAMICS

Final Dissertation

Graph Neural Networks for fluid mechanics: data-assimilation and optimization

---

by
Michele Quattromini:

Referees:

Prof. Taraneh Sayadi

Prof. Miguel A. Mendez

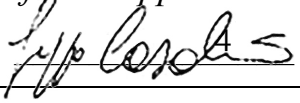
Supervisors:

Prof. Stefania Cherubini

Prof. Caroline Nore

Coordinator of Ph.D Program:

Prof. Giuseppe Casalino



Course n°37, 01/11/2021-31/10/2024

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences mécaniques et énergétiques, matériaux et géosciences (SMEMAG)

Titre: Graph Neural Networks pour la mécanique des fluides : data-assimilation et optimisation.
Mots clés: Apprentissage automatique, Apprentissage automatique informé par la physique, Graph Neural Networks, Dynamique des fluides numérique, Équations de Reynolds-Averaged Navier Stokes, Assimilation de données.

Résumé: Cette thèse de doctorat explore l'application des réseaux de neurones en graphes (GNN) dans le domaine de la dynamique des fluides numérique (CFD), avec un accent particulier sur l'assimilation de données et l'optimisation. Le travail est structuré en trois parties principales: assimilation de données pour les équations de Navier-Stokes moyennées à la Reynolds (RANS) basée sur des modèles GNN; assimilation de données augmentée par les GNN avec des contraintes physiques imposées par la méthode adjointe; optimisation des systèmes fluides par des techniques d'apprentissage automatique (ML).

Dans la première partie, la thèse examine le potentiel des GNN pour contourner les modèles de fermeture traditionnels, qui nécessitent souvent une calibration manuelle et sont sujets à des inexactitudes. En exploitant des données de simulation à haute fidélité, les GNN sont entraînés à apprendre directement les quantités non résolues de l'écoulement, offrant ainsi un cadre plus flexible pour le problème de fermeture des équations RANS. Cette approche élimine le besoin de modèles de fermeture calibrés manuellement, fournissant une alternative généralisée et basée sur les données. De plus, dans cette première partie, une étude approfondie de l'impact de la quantité de données sur les performances des GNN est réalisée, avec la conception d'une stratégie d'Active Learning pour sélectionner les données les plus informatives parmi celles disponibles.

Sur la base de ces résultats, la deuxième partie de la thèse aborde un défi critique souvent rencontré par les modèles d'apprentissage automatique: l'absence de garantie de cohérence physique dans leurs prédictions. Afin de garantir que les GNN non seulement minimisent les erreurs, mais produisent également des résultats physiquement valides, cette partie intègre des contraintes physiques directement dans le processus d'entraînement des GNN. En incorporant les principes clés de la mécanique des fluides dans le cadre de l'apprentissage automatique, le modèle produit des prédictions à la fois fiables et cohérentes avec les lois physiques sous-jacentes, améliorant ainsi son applicabilité aux problèmes réels.

Dans la troisième partie, la thèse démontre l'application des GNN pour optimiser les systèmes de dynamique des fluides, avec un accent particulier sur la conception des éoliennes. Ici, les GNN sont utilisés comme modèles de substitution, permettant des prédictions rapides de diverses configurations de conception sans avoir besoin de réaliser une simulation CFD complète à chaque itération. Cette approche accélère considérablement le processus de conception et montre le potentiel de l'optimisation basée sur l'apprentissage automatique dans le cadre de la CFD, permettant une exploration plus efficace des espaces de conception et une convergence plus rapide vers des solutions optimales.

Sur le plan méthodologique, la thèse introduit une architecture GNN sur mesure spécifiquement adaptée aux applications CFD. Contrairement aux réseaux de neurones traditionnels, les GNN sont intrinsèquement capables de gérer des données de maillage non structurées, ce qui est courant dans les problèmes de mécanique des fluides impliquant des géométries irrégulières et des domaines d'écoulement complexes. À cette fin, la thèse présente une interface en deux parties entre les solveurs de la méthode des éléments finis (FEM) et l'architecture GNN. Cette interface transforme les champs vectoriels FEM en tenseurs numériques pouvant être traités efficacement par le réseau neuronal, permettant ainsi l'échange de données entre l'environnement de simulation et le modèle d'apprentissage.

Title: Graph Neural Networks for fluid mechanics: data-assimilation and optimization
Keywords: Machine Learning, Physics-Informed Machine Learning, Graph Neural Networks, Computational Fluid Dynamics, Reynolds-Averaged Navier-Stokes, Data assimilation

Abstract: This PhD thesis investigates the application of Graph Neural Networks (GNNs) in the field of Computational Fluid Dynamics (CFD), with a focus on data-assimilation and optimization. The work is structured into three main parts: data-assimilation for Reynolds-Averaged Navier-Stokes (RANS) equations based on GNN models; data-assimilation augmented by GNN and adjoint-based enforced physical constraint; fluid systems optimization by ML techniques.

In the first part, the thesis explores the potential of GNNs to bypass traditional closure models, which often require manual calibration and are prone to inaccuracies. By leveraging high-fidelity simulation data, GNNs are trained to directly learn the unresolved flow quantities, offering a more flexible framework for the RANS closure problem. This approach eliminates the need for manually tuned closure models, providing a generalized and data-driven alternative. Moreover, in this first part, a comprehensive study of the impact of data quantity on GNN performance is conducted, designing an Active Learning strategy to select the most informative data among those available.

Building on these results, the second part of the thesis addresses a critical challenge often faced by ML models: the lack of guaranteed physical consistency in their predictions. To ensure that the GNNs not only minimize errors but also produce physically valid results, this part integrates physical constraints directly into the GNN training process. By embedding key fluid mechanics principles into the machine learning framework, the model produces predictions that are both reliable and consistent with the underlying physical laws, enhancing its applicability to real-world problems.

In the third part, the thesis demonstrates the application of GNNs to optimize fluid dynamics systems, with a particular focus on wind turbine design. Here, GNNs are employed as surrogate models, enabling rapid predictions of various design configurations without the need for performing a full CFD simulation at each iteration. This approach significantly accelerates the design process and demonstrates the potential of ML-driven optimization in CFD workflows, allowing for more efficient exploration of design spaces and faster convergence toward optimal solutions.

On the methodology side, the thesis introduces a custom GNN architecture specifically tailored for CFD applications. Unlike traditional neural networks, GNNs are inherently capable of handling unstructured mesh data, which is common in fluid mechanics problems involving irregular geometries and complex flow domains. To this end, the thesis presents a two-fold interface between Finite Element Method (FEM) solvers and the GNN architecture. This interface transforms FEM vector fields into numerical tensors that can be efficiently processed by the neural network, allowing data exchange between the simulation environment and the learning model.

Contents

Nomenclature

Relationships

$=$	Equality
$\coloneqq$	Definition
$\leftarrow$	Assignment

Vector and matrices

a	Scalar
$\mathbf{a}$	Vector
$\mathbf{A}$	Matrix
A_{ij}, A_i, A_j	(i,j)-entry, i-th row, j-th column of a matrix $\mathbf{A}$
$\mathbf{a}^T, \mathbf{A}^T$	Transpose of vector $\mathbf{a}$, transpose of matrix $\mathbf{A}$
$\|\mathbf{a}\|_p \coloneqq \left(\sum_{j=1}^{n} \|a_j\|^p\right)^{\frac{1}{p}}$	ℓ^p-norm of $\mathbf{a} \in \mathbb{R}^n$, ℓ^1 is the absolute norm, ℓ^2 is the Euclidean norm
$\langle \cdot, \cdot \rangle$	Inner product
$\mathbf{I}$	Identity matrix

Computational Fluid Dynamics

$\mathbf{u} = (u, v, w)$	Velocity vector
$\overline{\mathbf{u}} = (\overline{u}, \overline{v}, \overline{w})$	Mean velocity vector
$\mathbf{f} = (f_x, f_y, f_z)$	Reynolds forcing stress
Re	Reynolds number
ν, μ	Kinematic viscosity, dinamic viscosity
Ω	Computational domain

Machine Learning

$\mathbf{x}, \mathbf{y}, \hat{\mathbf{y}}$	Input, target and predicted output vectors
σ	Non linear activation function

$\boldsymbol{\theta}$	Neural Network's weight matrix

Other symbols

$H^n(\Omega)$	Sobolev space, order n, domain Ω

Acronyms

ADAM Adaptive Moment Estimation. 47

AI Artificial Intelligence. 15, 16, 50, 131

BET Blade Element Theory. 122, 128, 130

BFGS Broyden-Fletcher-Goldfarb-Shanno. 69

CFD Computational Fluid Dynamics. 17, 19–26, 32, 33, 36–38, 49, 56–59, 61, 63, 70, 71, 73, 77, 103, 104, 117, 120, 122, 126, 127, 139–141

CFL Courant-Friedrichs-Lewy. 38, 82

CNN Convolutional Neural Network. 19, 21, 49, 80

DAWT Diffuser-Augmented Wind Turbine. 117, 119, 120, 122–125, 140, 149

DDPG Deep Deterministic Policy Gradient. 131–134

DL Deep Learning. 16

DNS Direct Numerical Simulation. 20, 21, 29, 31, 32, 35, 105–107, 110, 139

DOF Degrees Of Freedom. 29

FDM Finite Difference Method. 26

FEM Finite Element Method. 26, 36, 37, 39–42, 80, 81, 100, 105, 106, 113

FVM Finite Volume Method. 26

GAN Generative Adversial Network. 19, 49

GNN Graph Neural Network. 21–24, 43, 49, 51, 54–63, 65, 71, 73, 77, 79–81, 86–96, 98–110, 112–114, 120–122, 125–128, 130, 136–141, 146, 147, 149, 150

HAWT Horizontal Axis Wind Turbines. 117

LES Large Eddy Simulation. 31, 80

MAE Mean Absolute Error. 97, 147

ML Machine Learning. 15–23, 35, 43, 71, 79, 103, 104, 114, 117, 139–141

MLP Multi-Layer Perceptron. 15, 16, 43, 45, 48, 55–60, 126, 127, 134, 145

MP Message Passing. 55–58, 60

MSE Mean Squared Error. 46, 97, 106

NN Neural Network. 15, 16, 18–23, 43–50, 54, 61, 79, 80, 100, 101, 103, 104, 120, 125–128, 134, 135, 137, 149

NSE Navier-Stokes Equations. 17, 20, 25–31, 36, 42, 75, 76, 79, 82

OWT Open-Rotor Wind Turbines. 117, 124, 149

PDE Partial Differential Equation. 25, 36, 39, 41, 42, 70–73, 81

PINN Physics-Informed Neural Network. 18, 103

POD Proper Orthogonal Decomposition. 18

PPO Proximal Policy Optimization. 131

RANS Reynolds Averaged Navier-Stokes. 17, 18, 20–24, 29–33, 65, 70–74, 76, 77, 79, 87, 100, 103–108, 112–114, 120, 122, 124, 126, 127, 139

ReLU Rectified Linear Unit. 44

RL Reinforcement Learning. 15, 19, 66, 121, 125, 126, 131–138, 141

RNN Recurrant Neural Networks. 21, 49, 101, 140

SGD Stochastic Gradient Descent. 46, 47, 68

TBNNs Tensor-Basis Neural Networks. 18

1 - Introduction

1.1 . Impact of Machine Learning (ML) on Scientific World

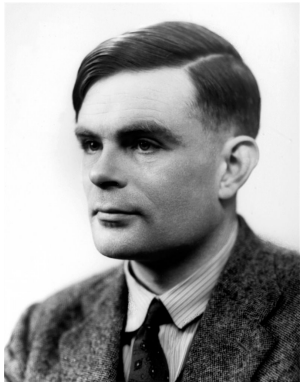


Figure 1.1: Alan Mathison Turing (London, 1912 – Wilmslow, 1954)

The history of ML dates back to mid-20th century. In 1950, Alan M. Turing (Fig. 1.1), a British mathematician and logician, posed the revolutionary question "Can machines think?" presented in his seminal paper "Computing Machinery and Intelligence" [Turing, 1950]. Turing questioned for the first time in history whether a machine could exhibit intelligent behavior comparable or indistinguishable from that of a human.

It's the birth of the modern *Machine Learning (ML)*. In less than 10 years, in the late 1950s, the term "Machine Learning" was officially coined by Arthur Lee Samuel, a USA professor and pioneer in Artificial Intelligence (AI) and Computer Sciences [Samuel, 1959]. Samuel's work, firstly on a checkers-playing game, effectively demonstrated that machines could be taught to improve their performance through experience, concept that, later on, will be known as Reinforcement Learning (RL), a branch of the broader AI.

During the next decade, in the 1960s, human brain's structure and functioning inspired the conception of the Neural Network (NN) whose first implementation was represented by the Perceptron idealized by Frank Rosenblatt, a USA psychologist [Rosenblatt, 1958].

From that point on and for more than 20 years, due to the strong critics by Marvin Minsky and Seymour Paper, the ML field falls in a forgotten area of study, known as *Artificial Intelligence winter*. In their book "Perceptron", indeed, they pointed out the insurmountable limitations of early NN as compared to the available computational resources of the time [Minsky and Papert, 1969].

The resurrection of the ML began between the 1980s and 1990s with the development, in 1986, of the back-propagation algorithm. Designed by David Rumelhart, Geoffrey Hinton and Ronald Williams, this algorithm soon became a milestone in the ML field [Rumelhart et al., 1986]. This mechanism, indeed, made it feasible to efficiently train NNs with more than one layers, known as Multi-Layer Perceptron (MLP). Accelerated by the huge increase of the computational power, research in ML finally gained significant momentum, and

techniques such as Decision Trees [Quinlan, 1986], Support Vector Machines [Cortes and Vapnik, 1995] and Ensemble Methods [Breiman, 1996] became prominent.
The 21st century has seen exponential growth in ML, driven by the advent of Big Data and the continuous increase of computational resources' availability. During this period, the concept of DL [Goodfellow et al., 2016] emerged, characterized by the use of NNs significantly deeper and novel architectures, which allowed for the modeling of complex patterns and hierarchical representations far beyond what traditional MLPs could achieve. This new subfield of ML has led to breakthroughs in fields such as image and speech recognition, natural language processing and autonomous systems.
The global attention on ML was further boosted by the victory in 2016 of the DeepMind's *AlphaGo*, a ML based software, which defeated the world champion Go player [Silver et al., 2016].
Since then, researcher throughout the world have contributed to ML advancement. More and more scientists began to question about the benefits of ML in scientific progress, leading to a huge dissemination of AI algorithms across almost every scientific discipline.
As a matter of fact, Machine Learning has profoundly transformed the scientific landscape, with its unparalleled capability to process and analyze vast sets of data, uncover complex patterns, generative and predictive potential.
In genomics, for example, analysis of large scale DNA with sequencing data ML techniques has paved the way to discoveries about genetic disorders, evolutionary biology and personalized treatments [Libbrecht and Noble, 2015].
On the medical point of view, among many other applications, it has been successfully used to enhance biomarkers analysis for early cancer diagnosis [Esteva et al., 2019].
In chemistry, ML models can be used to predict molecular properties and reactions, accelerating the process of drugs design and materials discovery [Butler et al., 2018].
In physics, ML algorithm can analyze experimental data to further enhance the physics phenomena understanding. The Large Hadron Collider (LHC) at CERN, for instance, heavily relies on ML to process and interpret the immense volumes of data produced by particle collisions [Radovic et al., 2018].
Therefore, ML has a deep impact on the scientific world by enhancing data-driven discovery, improving predictive models, accelerating research and fostering interdisciplinary collaborations. Nowadays, ML is still a very promising research field on itself and for its application to scientific world and its popularity and influence is expected to grow even further in the near future.

1.2 . Machine Learning (ML) in Computational Fluid Dynamics (CFD)

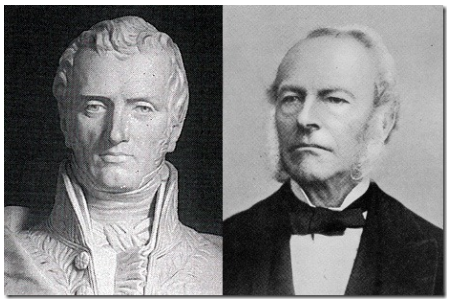


Figure 1.2: Claude-Louis Navier (Left) and George Stokes (Right)

Among the many scientific fields being revolutionized by the increasing integration of ML, Computational Fluid Dynamics (CFD) offers a very promising area for significant advancements.
CFD is a branch of fluid mechanics that uses numerical analysis to solve problems involving fluid flows. CFD spans through a wide range of applications including aerodynamics, weather forecasting, ocean motion and environmental engineering. By solving the Navier-Stokes Equations (NSE), proposed in the 19th century by Claude-Louis Navier and George Stokes (see Fig. 1.2), which describe the motion of fluid flows, CFD allows simulating the behavior of fluids in various conditions, providing insights that are often difficult or impossible to obtain experimentally. However, this process can be computationally expensive and time-consuming, and it often requires significant expertise to achieve accurate results. ML inclusion in traditional CFD techniques offers an innovative and promising approach to tackle these problems and enhance CFD's capability to explore a broader range of fluid dynamics phenomena, overcoming the limitations of mathematical traditional techniques [Brunton et al., 2020b]. ML models offer the capability to learn from large datasets of high-fidelity simulations or experimental data, enhancing the prediction of fluid behaviors. Brunton et al. [2020b] provides an extensive review of ML's applications to fluid mechanics, emphasizing how data-driven models can complement traditional methods and lead to enhanced simulations by reducing the need for empirical adjustments.
Alternative applications can be found for fluid mechanics in the realms of classification problems, clustering or control, where the number of contributions combining ML and standard techniques of analysis has been constantly growing in the last years [Raissi et al., 2020, Brunton et al., 2020a,b, Garnier et al., 2021, Vinuesa and Brunton, 2022, Mendez et al., 2023, 2022].

A major area where ML has been applied in CFD is turbulence modeling. Traditional approaches such as the Reynolds Averaged Navier-Stokes (RANS) equations, used for time-averaged flow quantities, introduce a closure term that needs to be modeled. This can be done using the classical closure models discussed in section 2.3.4, or leveraging ML for directly ap-

proximating the Reynolds stress or improving the existing models [Lapeyre et al., 2019, Beck and Kurz, 2021, Volpiani et al., 2021, Patel et al., 2024, Zhao et al., 2020]. Among the numerous authors that addressed this problem, Ling and Templeton [2015] applied classification methods for identifying regions of uncertainties where the closure term of the RANS might fail; Ströfer and Xiao [2021] combined data assimilation with NN modelling of the Reynolds stress using limited observation. Other approaches leverage baseline models such as the Spalart-Allmaras closure [Singh and Duraisamy, 2016], Physics-Informed Neural Network (PINN)s [Eivazi et al., 2022, Patel et al., 2024], random forests [Wang et al., 2017], regression methods [Schmelzer et al., 2020], decision trees [Duraisamy et al., 2019], ensemble methods [McConkey et al., 2022], genetic programming [Weatheritt and Sandberg, 2016, Zhao et al., 2020] or Bayesian approaches [Xiao et al., 2016]. Dupuy et al. [2023a] introduced a data-driven wall modeling approach for turbulent separated flows, demonstrating the potential of neural networks to model wall shear stress effectively, even in complex separated regions. For a broader overview, we refer to Duraisamy et al. [2019] and Beck and Kurz [2021], where the different levels of approximation are discussed together with a critical take on the limitations of the approach. In the specific case of an eddy viscosity closure model, the recent studies by Ling et al. [2016] demonstrated the effectiveness of Tensor-Basis Neural Networks (TBNNs) in learning a General Eddy Viscosity model type [Pope, 1975]. TBNNs capitalize on the tensor decomposition approach proposed by Pope to account for invariances and streamline the number of parameters to be learned. The inductive bias introduced by this modelling approach restricts TBNN application to nearly homogeneous flows with high Reynolds numbers, where local effects predominate [Cai et al., 2024].

In the context of dimensionality reduction, Beck and Kurz [2021] explored the use of ML for dimensionality reduction, combining these techniques with reduced-order models to create efficient solutions for high-dimensional fluid problems.
Among the ML applications, clustering and classification techniques are playing an important role in Fluid Mechanics, due to their versatility in identifying flow structures and complementing advanced methods, such as the multi-scale Proper Orthogonal Decomposition (POD) [Mendez et al., 2019]. These methods are used for identifying flow structures or - for the case of POD or related techniques - used as basis of projection for computing reduced order models preserving the key flow characteristics. Kaiser et al. [2014] successfully employed k-means clustering to discretize a high-dimensional phase space for the fluid mixing layer. This approach was later enhanced by Colvert et al. [2018], who used NNs to classify wake topologies behind a pitching airfoil. Their work highlights how ML can automate the identification of flow

features, improving both accuracy and efficiency.

Super-resolution, another area where ML has proven valuable, has been used to enhance the resolution of fluid flow simulations. Fukami et al. [2018] developed a CNN-based super-resolution algorithm to reconstruct turbulent flow fields, accurately preserving the energy spectrum. Xie et al. [2018] employed Generative Adversial Network (GAN)s to further improve the resolution of flow simulations, demonstrating that ML can provide high-fidelity results with reduced computational costs.

ML has also found extensive application in both the optimization and control of fluid systems. To begin with, we cite the comparative analysis by Pino et al. [2023], where various ML methods for active flow control are assessed. Among them, Rabault et al. [2019] used deep Reinforcement Learning (RL) to control oscillatory laminar flows; Bucci et al. [2019] applied Deep RL to control chaotic systems governed by the Kuramoto-Sivashinsky equation, which is a benchmark for spatiotemporal chaotic systems; Lee et al. [1997] investigates the use of NNs, for turbulence control aimed at drag reduction, specifically employing feedforward NNs trained to optimize control strategies; Colabrese et al. [2017] applied RL to optimize the movement of micro-swimmers, tiny artificial agents that navigate fluid environments; Sun et al. [2020] employed a ML algorithm as a surrogate model to rapidly predict metrics of fluid flow cases, without performing the entire CFD simulation. In this latter context, ML can, therefore, also enhance the spectrum of potential configurations by exploring a larger solution space and even find optimal solutions that may seem not intuitive for engineers.

Despite these benefits, there are some warnings to be aware of when integrating ML in CFD. Firstly, every ML algorithm is fueled by a huge availability of data [Jordan and Mitchell, 2015]. Moreover, high-quality data are necessary for an accurate NN training. Generating and maintaining such data can be resources demanding and computationally expensive, although without it, the potential of a ML based model is severely limited.
While ML algorithms are often valued for their potential to model complex patterns, their ability to generalize to unseen scenarios remains a significant challenge, particularly when the unseen data falls outside the range or vicinity of the training data. The threshold at which the model's capability to generalize starts to degrade is typically loose and not well-defined, making predictions beyond the training range less reliable and prone to inaccuracies. These latter limits, often related to the *overfitting* and *underfitting* problems, are of great concern and must be carefully taken into consideration to avoid poor prediction results.

Eventually, ML algorithms in their original conception, are fully data-driven as they leverage data to be trained, and they act as a statistical tool on these data. The physical plausibility of a NN outcome, therefore, is not a-priori granted. As a consequence, many ML algorithms require the incorporation of physical constraints during the training process to ensure coherence and reliability in their predictions. This process often involves sophisticated techniques that blend traditional physics-based methods with advanced ML approaches [Willard et al., 2020]. When used with caution, the application of ML algorithm to CFD field is, ultimately, leading this latter to a whole new level by improving the efficiency of classical techniques or reducing the computational expenses naturally associated with CFD simulations.

1.3 . The chosen physical model: introduction to Reynolds Averaged Navier-Stokes Equations (RANS)

In the realm of CFD, the selection of an appropriate physical model is essential to achieve a balance between computational feasibility and the desired level of accuracy. The most accurate available method is Direct Numerical Simulation (DNS) [Moin and Mahesh, 1998], which involves solving the full, time-dependent NSE without introducing any simplifying assumptions or modeling approximations. This method directly resolves all scales of turbulence (Sec. 2.3.1), capturing the full spectrum of fluid motion. However, despite the accuracy it offers, the computational cost associated with DNS is prohibitively high for most practical applications, especially in industrial settings.

In many industrial environments, indeed, the focus often shifts away from capturing the detailed, chaotic and transient fluid behavior to obtaining key integral quantities, such as forces on structures, heat transfer rates, or time averaged flow characteristics over the entire computational domain. These quantities are typically statistically steady or time-averaged, making a highly detailed resolution of turbulent structures unnecessary. For these applications, the trade-off between accuracy and computational efficiency becomes critical. This is where the RANS model excels. RANS simplifies the problem by focusing on time-averaged quantities, rather than attempting to resolve the full turbulent structure at every scale [Pope, 2000].

While this significantly reduces the computational demands, it introduces additional challenges: the turbulent fluctuations need to be modeled through turbulence closure schemes. These schemes provide approximations for the unknown terms introduced by the averaging process, but their accuracy can vary depending on the flow conditions. The empirical nature of many turbulence models involve correlations and assumptions that may not hold universally and as a result, careful calibration of the turbulence model to the specific

flow conditions is often required to ensure accurate results.
Nonetheless, for many industrial applications, where the primary concern is obtaining reliable predictions of averaged quantities at a reasonable computational cost, RANS remains the most widely used approach.
In conclusion, while DNS offers unmatched accuracy by capturing all scales of turbulence, its computational cost makes it impractical for most real-world applications, particularly when combined with machine learning (ML) algorithms. These latter, indeed, require large amounts of data and therefore a huge number of simulations. For these reasons, the RANS model has been chosen as the reference physical model in this thesis, as it allows for a feasible integration with ML algorithms.

1.4 . The chosen ML architecture: introduction to Graph Neural Network (GNN)

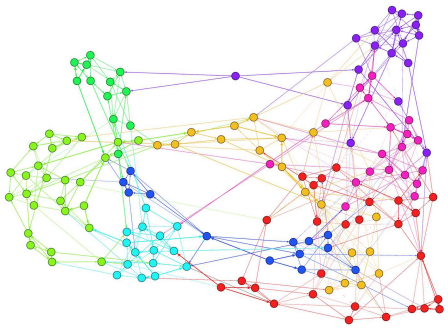


Figure 1.3: A Graph Neural Network (GNN) representation

In the realm of ML, selecting the NN architecture is critical to achieve optimal performances. While traditional NN such as Convolutional Neural Network (CNN)s [LeCun et al., 1998] or Recurrant Neural Networks (RNN)s [Hopfield, 1982] have demonstrated exceptional performance in processing structured data such as images or time series respectively, they face significant limitations when applied to unstructured data, which is often the case in CFD field. When dealing with complex geometries or intricate fluid flow cases, indeed, the refinement of the mesh is of fundamental importance to achieve accurate results in the simulation. Structured meshes, exhibiting regular and ordered elements to represent the computational domain, are sometimes not flexible enough to handle these cases; unstructured meshes are therefore essential and lead to great improvement in the simulation accuracy. Therefore, in CFD, data typically come from spatial discretization in the form of irregular, unstructured meshes.
To overcome these challenges, Graph Neural Network (GNN)s (Fig. 1.3) [Scarselli et al., 2008] have emerged as a powerful alternative. Unlike CNNs or RNNs, GNNs are explicitly designed to operate on graph-structured data, making them ideally suited for the kind of unstructured, relational data found in CFD simulations. In a GNN, nodes represent entities such as mesh points, while edges capture the interactions or dependencies between these entities,

mirroring the relationships found in fluid flow simulations mesh points. This feature improves the capability of this ML architecture to effectively capture the spatial dependencies between data, leading to a more accurate and realistic simulations of the fluid behavior.
Another key aspect of the GNNs is their inherent invariance to permutation [Scarselli et al., 2008] of the nodes in the graph, meaning that the NN's output does not depend on the order of the nodes. This feature is essential for the generalization capability since it allows the GNN to be completely detached from the position of the mesh points within the computational domain and therefore be able to provide predictions for geometric configurations never seen in the training process.
For all these reasons, the GNN architecture offers an excellent choice for CFD problems, as a robust and flexible learning framework. It will be the standard ML base when referring to NN architecture in this thesis.

1.5 . Objectives of the thesis

The main goal of this thesis is to investigate the extent to which a ML algorithm, specifically a GNN, can be effectively integrated into the framework of RANS simulations in the CFD context. This research focuses on evaluating various applications of GNNs within RANS simulations to demonstrate their potential advantages over traditional CFD techniques, particularly in terms of improving prediction accuracy and reducing computational costs.
The thesis is organized around three key challenges and objectives, each addressing a critical aspect of the integration of ML models into CFD workflows:

- Addressing the RANS Closure Problem: justified by the growing body of literature demonstrating the success of data-driven models in learning complex, nonlinear relationships [Wang et al., 2020, Pfaff et al., 2021], the first objective of this thesis is to explore whether GNNs can leverage high-fidelity simulation data, to directly infer the missing RANS closure term. This approach is focused on transitional flow regimes where the complexity associated with turbulence modeling is reduced, providing a proof-of-concept for this methodology. While not imposing additional physical constraints, this work explores the feasibility of data-driven closure modeling in a controlled setting, paving the way for future generalizations.

- Physical Consistency in Data-Driven Approaches: While purely data-driven models have shown great potential, they are not without limitations. A common issue with data-driven ML models is their tendency to produce physically inconsistent results. For instance, ML models may predict solutions that violate fundamental physical laws, such as conservation

of mass or momentum. The second objective of this thesis is to address this challenge by incorporating physical constraints directly into the training process of the GNN. This approach, that will be referred to as physics-constrained ML, involves embedding known physical equations into the architecture of the GNN to guide the learning process. By doing so, the model is not only trained to minimize error against the data, but also to ensure that its predictions adhere to key physical principles. This hybrid approach, combining data-driven learning with physics-based constraints, is expected to improve the robustness and reliability of the GNN's predictions, making it more suitable for real-world engineering applications where physical consistency is critical.

- Optimization in Fluid Systems: The third objective of this thesis is to test the proposed GNN architecture in the context of fluid system optimization, focusing on a practical case involving the cross-section shape optimization of a duct placed around a wind turbine rotor, with the aim of increasing its power output. The goal, here, is not solely to focus on the specifics of wind turbine design, but rather to show how GNNs can transform the optimization process in fluid dynamics applications. In conventional CFD-based optimization, a series of simulations are run iteratively with varying control parameters, to converge on an optimal solution. In this thesis, GNNs are introduced as a surrogate model to predict the outcomes of different design configurations without needing to run a full CFD simulation for each iteration.

1.6 . Structure of the thesis

The thesis is structured in two parts. In the first part, covering the chapters 2-4, we briefly introduce the theoretical foundations and the methodologies, including some practical applications of GNN in CFD. In particular, the first section set the stage providing basic concepts and nomenclature for the CFD (Sec. 2), including the underlying governing equations (Sec. 2.2), and basics on the numerical schemes adopted in the thesis. The section also includes a thorough explanation of the RANS equations (Sec. 2.3), turbulence modeling challenges (Sec. 2.3.1), and the RANS closure problem (Sec.2.3.3).
Following the CFD section, the thesis shifts into the theoretical framework of NNs (Sec. 3) before delving into the specific of the GNNs architecture. Sec. 3.7 is dedicated to provide an overview of this NN architecture, covering its key components (Sec. 3.7.1). The chapter also details the custom GNN designed in the context of this thesis (Sec. 3.8) and the training algorithms used for the training process (Sec. 3.8.2). Finally, the first part finalizes with an introduction to adjoint methods (Sec. 4), adopted to enforce physical constraints directly into the GNN training loop. In the specific, the adjoint method allows for the

efficient calculation of gradients, enabling the model to learn from both the data and the governing physical equations.
The second part consists of three distinct yet interconnected projects (see Sec. 1.5) demonstrating the practical application of GNNs to CFD problems. These projects serve as case studies to highlight the potential benefits of integrating GNNs into RANS simulations or fluid systems' optimization. The first two works in (Sec. 5 and Sec. 6) are adapted from submitted journal articles, while the third (Sec. 7) is a technical report.

2 - Computational Fluid Dynamics (CFD)

2.1 . General Aspect

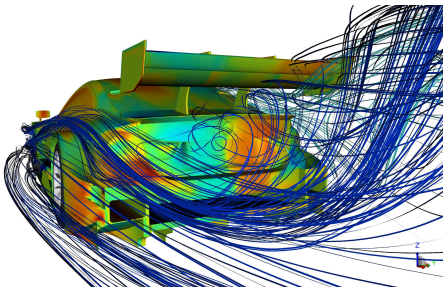


Figure 2.1: A fluid dynamic simulation

Computational Fluid Dynamics (CFD) is a branch of fluid mechanics that uses numerical methods and algorithms to analyze and solve problems that involve fluid flows (Fig. 2.1). The development of CFD as a discipline can be traced to the beginning of the 1960s with the advent of digital computers. At the very beginning, this discipline was focused on simple flow problems and laid the foundation for the sophisticated methods used today. It was during the 1970s that CFD gained its proper traction in scientific research. Over the decades, advances in computational power, numerical methods, and understanding of fluid dynamics have transformed CFD into a critical tool in both academic research and industrial applications.
The vast importance of CFD lies in its ability to provide detailed and comprehensive insight into fluid flow phenomena without the need for physical prototypes or experiments, which can be expensive and time-consuming. By using CFD, engineers and scientists can predict performance, identify potential issues and optimize designs in a virtual environment for fluid dynamics related applications.
Due to its broad range of applications, from aerospace to biomedical engineering, CFD is nowadays a fundamental tool in modern science.

2.2 . Navier-Stokes (NS) Equations

In CFD, the behavior of fluid flows is governed by mathematical equations that describe how quantities like velocity, pressure, and temperature evolve over space and time. The most critical of these equations for CFD are the Navier-Stokes Equations (NSE), which provide a detailed description of fluid flow evolution. The NSE are derived from the fundamental principles of conservation of mass, momentum and energy and are essential for accurately modeling fluid dynamics across a wide range of applications.
The NSE are a set of nonlinear Partial Differential Equation (PDE)s, particularly challenging to solve analytically. Interestingly, the NSE are also at the center

of one of the most famous unsolved problems in mathematics, highlighted by the *Millennium Prize Problems*.
Given the difficulty in finding exact closed solutions, the field of CFD has emerged as a practical and powerful alternative. By employing numerical methods, CFD enables the approximation of solutions to the NSE, making it possible to simulate complex fluid behaviors in a wide range of engineering and scientific applications. Instead of seeking exact analytical solutions, CFD breaks down the flow domain into discrete elements and applies numerical techniques to solve the governing equations locally. These numerical methods, such as the Finite Volume Method (FVM), the Finite Element Method (FEM), or the Finite Difference Method (FDM), allow for the discretization of the fluid domain, transforming the NSE into a system of algebraic equations that can be solved iteratively. By leveraging the power of computational resources, CFD has become indispensable for modeling fluid dynamics, enabling engineers and scientists to predict and analyze flow behaviors that would be otherwise impossible to capture analytically.

The NSE include the continuity equation (Eq. 2.1), which ensures the conservation of mass, the momentum equations (Eq. 2.2), derived from Newton's second law, and the energy equation, which accounts for the conservation of energy within the system. However, in the case of incompressible flows, which are the focus of this thesis, the energy equation can be neglected. This simplification is justified by the fact that in incompressible flows, changes in fluid density are negligible. Consequently, the variations in internal energy, which the energy equation would capture, are minimal and have little to no impact on the velocity and pressure fields of the flow [White, 2006].

2.2.1 . Continuity Equation

The equation ensuring the conservation of mass, also referred to as the continuity equation, for incompressible flows is expressed as

$$\nabla \cdot \mathbf{u} = 0, \tag{2.1}$$

where $\mathbf{u} = (u, v, w)$ is the velocity vector representing the fluid velocities in the x, y and z directions respectively. The physical meaning of this equation is that, for an incompressible flow, the volume of fluid entering and leaving any given control volume is balanced, ensuring that no accumulation or depletion of mass occurs over time. In other words, for incompressible flows, the density of the fluid remains constant, and thus the rate of mass across any arbitrary boundary must be conserved. Without satisfying this condition, fluid simulations would yield physically unrealistic results, such as the artificial creation or destruction of mass. Therefore, enforcing the continuity equation is a necessary step to ensure the accuracy and physical relevance of CFD simulations for incompressible flows.

2.2.2 . Momentum Equation

The conservation of momentum equation states that the rate of change of momentum within a fluid system must be balanced by the forces acting of the fluid. In other words, the momentum of a fluid element (the product of mass and velocity) changes over time due to the transport of momentum (through convection) and the influence of external forces. This fundamental concept is mathematically represented by the momentum equation, which for an incompressible Newtonian fluid takes the form:

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla p + \mu\nabla^2\mathbf{u} + \mathbf{F}. \tag{2.2}$$

Each term in this equation has a specific physical meaning, contributing to the description of how forces and flow dynamics interact within the fluid:

- Density (ρ): represent the fluid's density, assumed to be constant for incompressible flows.
- Local Acceleration $\left(\frac{\partial \mathbf{u}}{\partial t}\right)$: captures the rate of change of the fluid velocity at a specific point in space over time. In steady-state conditions, this term becomes zero, simplifying the momentum equation further.
- Convective Acceleration $(\mathbf{u}\cdot\nabla\mathbf{u})$: describes how the velocity changes due to the movement of the fluid itself. It represents the transport of momentum by the flow, meaning that as fluid particles move from one region to another, they carry their momentum with them.
- Pressure Gradient Force $(-\nabla p)$: where p is the pressure field. This term reflects the force (per unit area) exerted by pressure differences within the fluid. Fluids naturally move from regions of higher pressure to regions of lower pressure, and this term drives much of the motion in many fluid flows. In physical terms, the pressure gradient force represents the driving force in fluid flows phenomena.
- Viscous Force $(\mu\nabla^2\mathbf{u})$: with μ being the dynamic viscosity. This term accounts for the internal friction within the fluid due to viscosity, which acts to resist motion and causes the diffusion of momentum.
- External Forces $(\mathbf{F})$: represents any additional forces acting on the fluid from external sources, such as gravity, electromagnetic fields, or surface tension, which can influence the flow dynamics.

2.2.3 . Non-dimensionalization and Dimensionless Equations

Often the NSE are presented in their non-dimensional form. This process, known as non-dimensionalization, is a mathematical technique used to reduce the number of parameters governing the physical problem. By transforming the dimensional variables into their dimensionless counterpart, the

Scaling Parameter	Description	Dimensionless variable
U	Characteristic Speed	$\mathbf{u}^* = \dfrac{\mathbf{u}}{U}$
L	Characteristic Length	$\mathbf{x}^* = \dfrac{\mathbf{x}}{L}$ and $\nabla^* = L\nabla$
L/U	Characteristic Time	$t^* = \dfrac{t}{L/U} = \dfrac{tU}{L}$
$P = \rho U^2$	Characteristic Pressure	$p^* = \dfrac{p}{\rho U^2}$

Table 2.1: Dimensionless Variable

equations are simplified and reveal dominant mechanisms governing the flow behavior. This approach allows for a clearer comparison of different flow regimes and systems, regardless of the specific physical units being used.

To begin with, it is necessary to identify the relevant dimensional variables; in incompressible flows, they are usually the velocity $\mathbf{u} = (u, v, w)$, the spatial coordinates $\mathbf{x} = (x, y, z)$, time t and pressure p. Next, we introduce characteristic scales to normalize these variables. The characteristic scales represent typical magnitudes of the relevant physical quantities, chosen based on the specifics of the problem or the geometry being studied. The typical characteristic scales, summarized in Tab. 2.1, are:

- Characteristic speed U: A representative velocity scale, such as the free-stream velocity.
- Characteristic length L: A typical length scale, such as the diameter of a bluff body or the length of an airfoil.
- Characteristic pressure P: A representative pressure scale, often taken as ρU^2, where ρ is the fluid density.

Plugging these dimensionless variables into the dimensional NSE and dividing by U^2/L, the resulting system of equations reads as:

$$\frac{\partial \mathbf{u}^*}{\partial t} + (\mathbf{u}^* \cdot \nabla^*)\,\mathbf{u}^* = -\nabla^* p^* + \frac{\nu}{UL}\nabla^{*2}\mathbf{u}^* + \mathbf{F} \tag{2.3a}$$

$$\nabla^* \cdot \mathbf{u}^* = 0 \tag{2.3b}$$

where ν is the dynamic viscosity. To further simplify, a dimensionless number can be introduced and referred to as the Reynolds Number (Re)

$$Re = \frac{\rho U L}{\mu} = \frac{UL}{\nu}. \tag{2.4}$$

The Reynolds number Re is an extremely important dimensionless parameter that quantifies the relative importance of the inertial forces (which drive the flow) to viscous forces (which resist the flow). The Reynolds number allows characterizing different flow regimes

- Low Reynolds numbers indicate that viscous forces dominate, leading to smooth, orderly, and laminar flow.
- Intermediate Reynolds numbers suggest a transitional regime, where the flow may shift from laminar to turbulent depending on the conditions.
- High Reynolds numbers signify that inertial forces dominate, leading to chaotic and turbulent flow with significant mixing and irregular fluid motion.

Critical Reynolds numbers characterize the threshold identifying the shifting between the different regimes, and can be determined according to the cases under investigations. By introducing the Reynolds number into the non-dimensional NSE (Eq. 2.3b), results in the final dimensionless formulation of the NSE:

$$\frac{\partial \mathbf{u}^*}{\partial t} + (\mathbf{u}^* \cdot \nabla^*)\,\mathbf{u}^* = -\nabla^* p^* + \frac{1}{Re}\nabla^{*2}\mathbf{u}^* + \mathbf{F} \tag{2.5a}$$

$$\nabla^* \cdot \mathbf{u}^* = 0. \tag{2.5b}$$

In the following, the * superscript will be dropped, and the NSE will be always expressed in their nondimensional form.

2.3 . Reynolds Averaged Navier-Stokes (RANS) Equations

Solving the complete NSE (Eq. 2.5b) for turbulent flows via DNS can be computationally prohibitive, especially for large-scale, high Reynolds number flows. The number of Degrees Of Freedom (DOF), or grid points, necessary for a DNS increases dramatically as the Reynolds number grows. Specifically, the computational cost of DNS scales with the Reynolds number Re as $Re^{9/4}$ [Pope, 2000]. This scaling is due to the fact that the smallest turbulent scales decrease in size as Re increases, which means that finer spatial and temporal resolutions are required to accurately capture the entire range of turbulent motions. As a result, for high Reynolds number flows, which are common in industrial and engineering applications, the number of grid points and time steps needed to solve the flow can become prohibitively large. For example, a DNS simulation of turbulent air flow around a large aircraft at $Re \sim 10^6$ would require a number of grid points on the order of $Re^{9/4} \sim 10^{15}$. Simulating such flows using DNS would be computationally infeasible with current technology. Therefore, various approximation methods have been developed to balance accuracy and computational feasibility.
The RANS approach is one of the most widely used models in industrial applications. This allows for capturing essential flow features without resolving every turbulent scale, making it suitable for practical engineering problems.

Although this thesis does not focus on turbulence models or turbulent flow cases, a brief introduction to turbulence is provided for completeness. This general overview provides the necessary context to better understand how the RANS equations are derived, and highlights the key differences between RANS and the full NSE.

2.3.1 . Introduction to turbulence

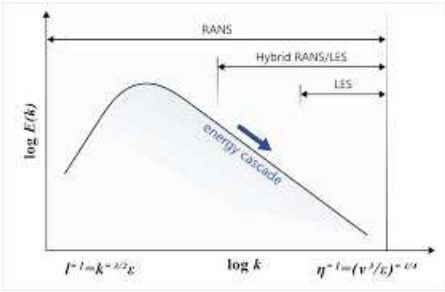


Figure 2.2: The Kolmogorov Energy Cascade

Turbulence is a highly complex, multiscale, chaotic phenomenon that arises in fluid flows when the Reynolds number (Eq. 2.4) is sufficiently high and inertial forces dominate over viscous forces. In turbulent flows, there is a wide range of energy scales, from large energy-containing eddies to small dissipative eddies where energy is dissipated as heat due to viscous forces. The Kolmogorov Energy Cascade (Fig. 2.2) better introduces this key concept. Introduced by Andrey Kolmogorov in 1941, this model explains how energy is transferred from large eddies down to progressively smaller eddies through a cascade process until it reaches the smallest scales, called the Kolmogorov scales, where energy dissipation occurs. The size of the Kolmogorov scale can be estimated as

$$\eta = \left(\frac{\nu^3}{\epsilon}\right)^{0.25}, \tag{2.6}$$

where ν is the kinematic viscosity and ϵ is the rate of energy dissipation. In other words, the largest eddies collapse and split into smaller eddies, transferring energy to finer scales until it eventually dissipates at the Kolmogorov scale.

For this reason, resolving all turbulent scales, from largest to Kolmogorov scale, requires fine meshes and appropriate time discretization, which is infeasible for most real world applications. Therefore, models are needed to approximate turbulent behavior without solving every scale explicitly.

2.3.2 . Approximate Models for Navier-Stokes Equations

To handle the challenges posed by turbulence, several approximation models have been developed, each targeting different scales of the turbulence spectrum. The primary approaches used in relation to the Kolmogorov energy cascade (Fig. 2.2) are:

- Direct Numerical Simulation (DNS): DNS resolves all turbulence scales by solving the full NSE without any approximation. While DNS provides highly accurate results, it is computationally prohibitive because it requires resolving the smallest Kolmogorov scales, demanding extremely fine spatial and temporal discretization.

- Large Eddy Simulation (LES): LES resolves the large, energy-containing eddies while modeling the smaller dissipative eddies. By resolving only the large-scale structures, LES reduces computational cost compared to DNS, but it is still computational demanding, particularly for high Reynolds number flows or complex geometries. The smallest scales are approximate using models like the Smagorinsky model [Pope, 2000].

- Reynolds Averaged Navier-Stokes (RANS): RANS completely models the turbulent phenomenon without resolving any of its scale. By time averaging the fluctuating quantities, this model focuses on the mean flow. While less accurate than DNS or LES, RANS is computationally efficient and widely used in cases where steady-state or time-averaged quantities are of interest, such as for example drag or lift. To approximate the effects of turbulence on the mean flow, a turbulence closure models is required. Some of the most commonly used includes the $k-\epsilon$ or the $k-\omega$ models as well as the Spalart-Allmaras model (Sec. 2.3.4) [Wilcox, 2006].

- Hybrid RANS/LES: Hybrid RANS/LES models combine the advantages of both RANS and LES approaches. The idea is to use RANS to model the flow in regions where turbulence is less significant and the mean flow dominates, while applying LES in regions where turbulent structures need to be resolved, such as near walls or in separated flows. This hybrid approach offers a compromise, improving accuracy in critical flow regions while keeping computational costs lower than a full LES simulation.

2.3.3 . From NS to RANS

The transition from the full NSE to the RANS formulation is achieved through Reynolds decomposition, which separates flow variables into mean and fluctuating components. Given the velocity field $\mathbf{u}(\mathbf{x},t) = (u,v,w)^T$, it can be decomposed as:

$$\mathbf{u}(\mathbf{x},t) = \overline{\mathbf{u}}(\mathbf{x}) + \mathbf{u}'(\mathbf{x},t) \tag{2.7}$$

where $\overline{\mathbf{u}}(\mathbf{x}) = (\overline{u},\overline{v},\overline{w})^T$ represents the time-averaged component of the unsteady velocity and $\mathbf{u}'(\mathbf{x},t) = (u',v',w')^T$ is the fluctuating component around the mean flow. Plugging the Reynolds decomposition (Eq. 2.7) into the NS

equations (Eq. 2.5b) and ensamble averaging results in:

$$\overline{\mathbf{u}} \cdot \nabla \overline{\mathbf{u}} + \nabla \overline{p} - \frac{1}{Re} \nabla^2 \overline{\mathbf{u}} = \mathbf{f} \tag{2.8a}$$

$$\nabla \cdot \overline{\mathbf{u}} = 0, \tag{2.8b}$$

where $\overline{p}$ is the mean pressure field and the term $\mathbf{f}$ is the Reynolds stress tensor. The introduction of the Reynolds stresses brings additional unknowns to the system, making the equations underdetermined. To close the system, turbulence closure models are used, the most common of which are detailed in Sec. 2.3.4, which approximate the Reynolds stresses based on known quantities. Ideally, $\mathbf{f}$ can be directly computed, when data are available, as:

$$\mathbf{f} = -\nabla \cdot (\overline{\mathbf{u}'\mathbf{u}'}) \tag{2.9}$$

In practice, mathematically computing $\mathbf{f}$ requires sufficient statistical convergence of the second-order statistics, which can be achieved through either DNS or experimental measurements. While time-resolved data can provide detailed insights, techniques such as Laser Doppler Velocimetry (LDV), which provide pointwise measurements, or Particle Image Velocimetry (PIV), which provide spatially resolved data, are often sufficient for obtaining reliable statistical averages. This challenge is commonly referred to in CFD as the RANS closure problem.

2.3.4 . RANS Turbulence Closure Models

This section provides a concise overview of the most widely used turbulence closure models for RANS equations. The theoretical foundations and equations presented here are derived primarily from Wilcox [2006], a comprehensive reference on turbulence modeling. The Reynolds stress tensor, $\mathbf{f}$ (Eq. 2.9), which appears in the RANS formulation, introduces additional unknowns. A turbulence closure model provides practical, computationally efficient approximations for these terms. Among the many turbulence closure models exiting in the literature for engineering applications, the most widely used are the k-ϵ model, the k-ω model and the Spalart-Allmaras model. Each of these models is fundamentally based on the Boussinesq approximation which models the Reynolds stresses as:

$$-\overline{\mathbf{u}'\mathbf{u}'} = \nu_t (\nabla \overline{\mathbf{u}} + \nabla \overline{\mathbf{u}}^T) - \frac{2}{3} k \mathbf{I} \tag{2.10}$$

where ν_t is the turbulent or eddy viscosity, k is the turbulent kinetic energy contributing to the isotropic component of the stress, and $\mathbf{I}$ is the identity matrix, ensuring that k contributes only to the normal (diagonal) components. It is important to note that the Spalart-Allmaras model omits the term $-\frac{2}{3} k \mathbf{I}$, due to its formulation, which does not explicitly depend on turbulent kinetic

energy. Consequently, it employs the resulting simplified expression for the Reynolds stresses. A short description of the most used models (non-exhaustive list) is provided below:

- k-ϵ model: it is one of the most widely used two-equation models in CFD for RANS-based turbulence modeling. It introduces two transport equations, one for the turbulent kinetic energy (k) and another for the rate of turbulent dissipation (ϵ). The transport equation for the turbulent kinetic energy k is formulated as:

$$\frac{\partial k}{\partial t} + (\overline{\mathbf{u}} \cdot \nabla)k = P_k - \epsilon + \nabla \cdot \left(\left(\nu + \frac{\nu_t}{\sigma_k} \right) \nabla k \right) \tag{2.11}$$

  where ϵ is the turbulent dissipation rate, σ_k is a model constant controlling the diffusion of k, ν_t represents the turbulent viscosity, whose expression will be provided in Eq. 2.14 below, and P_k is the turbulent kinetic energy production term expressed as:

$$P_k = \nu_t (\nabla \overline{\mathbf{u}} : \nabla \overline{\mathbf{u}}) \tag{2.12}$$

  The transport equation for the turbulent dissipation rate ϵ is given by:

$$\frac{\partial \epsilon}{\partial t} + (\overline{\mathbf{u}} \cdot \nabla)\epsilon = C_{\epsilon_1} \frac{\epsilon}{k} P_k - C_{\epsilon_2} \frac{\epsilon^2}{k} + \nabla \cdot \left(\left(\nu + \frac{\nu_t}{\sigma_\epsilon} \right) \nabla \epsilon \right) \tag{2.13}$$

  where σ_ϵ is a model constant for the diffusion of ϵ and C_{ϵ_1} and C_{ϵ_2} are empirically derived constants. The turbulent viscosity ν_t is computed as:

$$\nu_t = C_\mu \frac{k^2}{\epsilon} \tag{2.14}$$

  where C_μ is a model constant. The standard k-ϵ model is based on the following constants [Launder and Sharma, 1974]:

$$C_{\epsilon 1} = 1.44, \quad C_{\epsilon 2} = 1.92, \quad C_\mu = 0.09, \quad \sigma_k = 1.0, \quad \sigma_\epsilon = 1.3 \tag{2.15}$$

  The k-ϵ model is widely used due to its computational efficiency and versatility, particularly in free-shear flows such as jets and mixing layers. However, a notable disadvantage is its limited accuracy near walls, especially in flows with strong pressure gradients, separations, or adverse wall effects. The model also assumes isotropy in turbulence, which limits its accuracy in flows with high anisotropy, such as swirling or complex separated flows.

- k-ω model: another widely used two-equations turbulence model, similar in the structure to the previous k-ϵ model. This approach often enhances performance in boundary layers and flows with adverse pressure gradients. The transport equation for k in the k-ω model is formulated as:

$$\frac{\partial k}{\partial t} + (\overline{\mathbf{u}} \cdot \nabla)k = P_k - \beta^* k\omega + \nabla \cdot \left(\left(\nu + \sigma^* \frac{k}{\omega} \right) \nabla k \right) \tag{2.16}$$

where P_k is the kinetic energy production term as defined in Eq. 2.12, β^* and σ^* are model constants.
The transport equation for the specific dissipation rate ω is given by:

$$\frac{\partial \omega}{\partial t} + (\overline{\mathbf{u}} \cdot \nabla)\omega = \alpha \frac{\omega}{k} P_k - \beta \omega^2 + \frac{\sigma_d}{\omega} \nabla k \nabla \omega + \nabla \cdot \left(\left(\nu + \sigma \frac{k}{\omega} \right) \nabla \omega \right) \quad (2.17)$$

where α, β, σ_d and σ are model constants controlling the balance between production and dissipation of ω. With these equations, the eddy viscosity ν_t is calculated as:

$$\nu_t = \frac{k}{\omega} \quad (2.18)$$

The k-ω model is effective in simulating flows with strong adverse pressure gradients and boundary layer effects, which makes it a preferred choice for applications involving near-wall regions and complex boundary layer behaviors. However, this model can be sensitive to free-stream boundary conditions, which may lead to inaccuracies in flows where boundary conditions are uncertain or vary significantly.

- Spalart-Allmaras model: is a one-equation turbulence model. Unlike the k-ϵ and k-ω models, the Spalart-Allmaras model uses a single transport equation for a modified eddy viscosity $\tilde{\nu}$. This design simplifies the model, which is effective for boundary-layer-dominated flows. The transport equation for $\tilde{\nu}$ is given by:

$$\begin{aligned} \frac{\partial \tilde{\nu}}{\partial t} + (\overline{\mathbf{u}} \cdot \nabla)\tilde{\nu} = C_{b1}\tilde{S}\tilde{\nu} + \frac{1}{\sigma}\nabla \cdot ((\nu + \tilde{\nu})\nabla\tilde{\nu}) \\ + \frac{C_{b2}}{\sigma}(\nabla\tilde{\nu}) \cdot (\nabla\tilde{\nu}) - (C_{w1} f_w) \frac{\tilde{\nu}^2}{d^2} \end{aligned} \quad (2.19)$$

where d is the distance to the nearest surface and σ, C_{b1}, C_{b2}, $C_{\omega 1}$ are model constants. The function f_ω depends on the model constant while $\tilde{S}$ is defined as:

$$\tilde{S} = S + \frac{\tilde{\nu}}{k^2 d^2} f_{v2} \quad (2.20)$$

f_{v2} being a function of the model constants and S being the magnitude of the mean strain rate, defined as:

$$S = \sqrt{2\Omega_{ij}\Omega_{ij}} \quad (2.21)$$

with Ω_{ij} as the rotation tensor.
The eddy viscosity is then determined as:

$$\nu_t = \tilde{\nu} f_{v1} \quad (2.22)$$

where f_{v1} is a function designed to adjusts $\tilde{\nu}$ in near walls regions, improving performance in boundary layers. The Spalart-Allmaras model,

as a one-equation model, is computationally simpler and more efficient than two-equation models. However, the model is limited in accurately capturing complex turbulent flows with separation, reattachment, or high anisotropy, as it lacks a term explicitly representing turbulent kinetic energy.

In contrast to the turbulence closure models discussed above, this thesis employs a fundamentally different approach by directly deriving the Reynolds stresses $\mathbf{f}$ from their analytical formulation (Eq. 2.9), as obtained from the fluctuating components in DNS data. Unlike turbulence models, which impose predefined assumptions on turbulence structure and behavior, this method seeks to represent the turbulent stress tensor using information directly resolved by DNS, reducing reliance on potentially restrictive or inaccurate assumptions. While this method avoids the explicit modelling assumptions inherent to traditional turbulence closure models, the computation of Reynolds stresses from DNS data requires careful consideration of statistical convergence to ensure accurate representation of the turbulent stress tensor. This thesis aims to mitigate these limitations by combining high-fidelity DNS data with machine learning techniques that are capable of capturing complex, nonlinear dependencies within the data. The goal is to train ML models that are not only informed by accurate data derived from DNS but also generalizable to new, unseen flow conditions beyond those explicitly simulated. By minimizing explicit bias in the initial formulation of the Reynolds stress tensor, the ML model is provided with the flexibility to explore a broader range of potential relationships within the data. However, this flexibility is contingent upon the statistical reliability of the DNS-derived stresses, which requires rigorous preprocessing and validation to ensure consistency. The resulting model, informed by high-fidelity data and guided by machine learning inference, retains the full complexity of turbulent structures while aiming to generalize beyond the specific conditions of the original dataset.

2.4 . Finite Element Methods (FEM)

2.4.1 . Introduction

Figure 2.3: Boris Grigoryevich Galerkin (Polotsk, 1871 – Moscow, 1945)

Finite Element Method (FEM) is a numerical technique developed to find approximate solutions to complex PDEs in various engineering and scientific disciplines [Logg et al., 2012]. Its original formulation can be traced to the 1940s and 1950s, when it was primarily developed for structural analysis in aerospace engineering. The method gained widespread adoption in the 1970s due to advances in digital computing and numerical methods.
In CFD, FEM is used to numerically solve the NSE and other governing equations of fluid dynamics. The core strength of the FEM method is in its flexibility in handling complex geometries and fluid dynamic configurations.

The application of FEM typically follows a structured process that includes the following key steps:

- Spatial and Temporal Discretization: The first step is to discretize both the spatial and temporal domains. The physical domain is divided into finite discrete elements that cover the problem space, known as *mesh*. In case of time-dependent problems, the simulation time is also divided into discrete time steps, enabling the solution to evolve incrementally over time.

- Weak Formulation: The governing PDEs are transformed into their weak form, typically through the Galerkin (Fig. 2.3) method. The weak form reduces the complexity of the equations, making them suitable for numerical solutions.

- Assembly of Global System of Equations: The local equations from each element are then assembled into a global system of algebraic equations that represents the entire problem domain.

- Boundary Conditions: Applying appropriate boundary conditions is crucial to ensure a physically meaningful solution. These conditions can include prescribed velocities or pressures at the domain boundaries, and their correct implementation significantly affects the accuracy of the simulation.

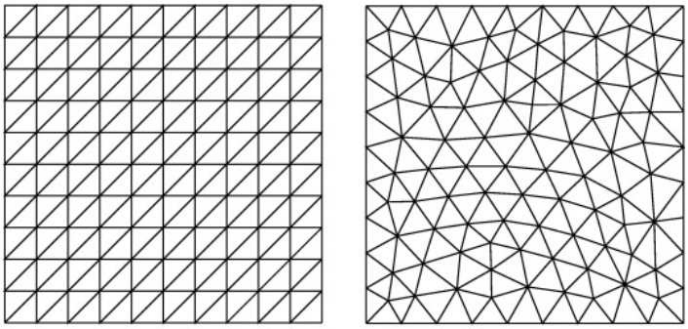


Figure 2.4: (Left) Structured mesh (Right) Unstructured mesh

- Numerical Solver: Finally, the assembled system of equations is solved using iterative numerical techniques. Given the large size of the systems involved in CFD, specialized solvers are used to efficiently handle the sparse matrices resulting from the FEM discretization.

2.4.2 . Spatial and Temporal Discretization

Spatial discretization refers to subdividing the physical domain into smaller, discrete elements, resulting in what is known as the *mesh*. The quality and type of spatial discretization are crucial, as they directly influence the accuracy and computational efficiency of the CFD simulation. The two primary types of meshes are:

- Structured Meshes (Fig. 2.4, Left): These are characterized by a regular, grid-like arrangement of elements, where each element (triangular, quadrilateral or hexahedral) follows a predictable and ordered pattern. The discretized domain can therefore be described by a uniform Cartesian grid:

$$x_i = i \cdot \Delta x, \quad y_j = j \cdot \Delta j, \tag{2.23}$$

  with i, j being the index of the node and Δx and Δy the element size in x and y direction, respectively. This regularity allows for efficient implementation, but structured meshes are less flexible in handling complex geometries or localized refinement, which is often required in CFD.

- Unstructured Meshes (Fig. 2.4, Right): These meshes are composed of elements arranged in an irregular pattern. The flexibility of unstructured meshes makes them ideal for handling complex geometries, as they can easily adapt to curved surfaces, sharp corners, and varying levels of detail. The flexibility of unstructured meshes comes at the cost of increased computational complexity and a more challenging solver implementation.

The choice between these two types of meshes hugely depends on the specific requirements and necessity of the CFD simulation.

In addition to discretizing space, temporal discretization is necessary for time-dependent problems. This process involves breaking the time domain into discrete intervals, called time steps, and approximating the time derivatives in the governing equations using finite difference methods. There are several commonly used methods for temporal discretization:

- Explicit Methods: In an explicit method, such as the forward Euler method, the time derivative is approximated as:

$$\frac{\partial \mathbf{u}}{\partial t} \approx \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} \tag{2.24}$$

  where n represents the time step. The future state $\mathbf{u}^{n+1}$ is computed directly from the known value at the current time step $\mathbf{u}^n$. These methods are simple to implement, but they are conditionally stable. The time step size must, indeed, be sufficiently small to ensure that information does not travel more than one grid cell in a single time step. This condition is known as Courant-Friedrichs-Lewy (CFL) condition and for the stability of explicit methods must be satisfied as:

$$\text{CFL} = \frac{U \cdot \Delta t}{\Delta x} \leq 1 \tag{2.25}$$

  where U is the characteristic velocity, Δx is the spatial mesh size, and Δt is the time step size. For problems with rapid changes or high velocities, this can lead to an impractically small time step, increasing the computational cost.

- Implicit Methods: In an implicit method, such as the backward Euler method, the time derivative is approximated as:

$$\frac{\partial \mathbf{u}}{\partial t} \approx \frac{\mathbf{u}^{n} - \mathbf{u}^{n-1}}{\Delta t} \tag{2.26}$$

  where n represents the time step. While implicit methods are computationally more expensive, they are unconditionally stable, allowing for larger time steps without risking instability.

- Crank-Nicolson Methods: This method is a combination of explicit and implicit methods. It averages the forward Euler and backward Euler methods to achieve second-order accuracy in time and it balances the accuracy of explicit methods with the stability of implicit methods

The coupling between spatial and temporal discretization impacts the stability and convergence of the simulation [Drazin and Reid, 2002, Charru, 2011].

For instance, finer meshes in regions of interest (e.g., boundary layers or areas with steep gradients) necessitate smaller time steps in explicit methods, increasing the overall computational cost. Implicit methods, though computationally intensive, allow for larger time steps, making them better suited for simulations where long-term accuracy is prioritized over computational speed.

2.4.3 . Weak Formulation

Named after Boris G. Galerkin (Fig. 2.3), a Russian mathematician, the Galerkin method is a key component of the FEM since it allows to convert a PDE into a system of algebraic equations that can be solved numerically. The essence of the Galerkin method lies in approximating the solution of a PDE as a linear combination of basis functions, transforming a continuous problem into a discrete one.

Consider a PDE such as the Poisson equation, which often arises in physics and engineering problems. The equation is given by

$$-\nabla^2 \mathbf{u} = \mathbf{s}, \tag{2.27}$$

where $\mathbf{u}$ is the unknown solution and $\mathbf{s}$ is a source term.

The first step in FEM consists of deriving a weak form (or variational form) of the PDE. This is done by multiplying the governing equation by a test function $\mathbf{v}$, chosen from a suitable function space, and integrating over the computational domain Ω

$$-\int_{\Omega} \mathbf{v}\nabla^2 \mathbf{u} d\Omega = \int_{\Omega} \mathbf{v}\mathbf{s} d\Omega. \tag{2.28}$$

The test function $\mathbf{v}$ is typically chosen from the same function space as the trial (or solution) function $\mathbf{u}$. The appropriate function space is typically the Sobolev space, $H^1(\Omega)$, which consists of functions that are square-integrable along with their first derivatives. Mathematically, this is expressed as

$$H^1(\Omega) = \left\{ \mathbf{u} \in L^2(\Omega) | \nabla \mathbf{u} \in L^2(\Omega) \right\}, \tag{2.29}$$

which ensures that $\mathbf{u}$ and $\nabla \mathbf{u}$ are square-integrable on the computational domain and thus that the integral is well-defined and meaningful in the weak form. To reduce the complexity of the equation and lowering the order of differentiation required, an integration by parts is then applied. Applying this technique to the weak form of the Poisson equation leads to

$$\int_{\Omega} \nabla \mathbf{v} \cdot \nabla \mathbf{u} d\Omega - \int_{\partial\Omega} \mathbf{v} \nabla \mathbf{u} \cdot \mathbf{n} d\mathrm{T} = \int_{\Omega} \mathbf{v}\mathbf{s} d\Omega, \tag{2.30}$$

where $\partial\Omega$ is the boundary of the domain Ω and $\mathbf{n}$ is the outward normal to the boundary. This latter formulation involves quantities defined on the boundary of the domain, therefore incorporating appropriate boundary conditions

is crucial as they are essential in ensuring that the solution behaves as expected at the edges of the domain (Sec. 2.4.5).

Once the weak form has been established, the next step is to approximate the solution $\mathbf{u}$ and the test function $\mathbf{v}$ using basis functions (also called shape functions). These functions are defined locally on each element of the discretized mesh and represent the solution in terms of its values at the mesh nodes. The finite element approximation of the solution and test functions can be written as

$$\mathbf{u}_h = \sum_{i=1}^{N} U_i \phi_i, \qquad \mathbf{v}_h = \sum_{i=1}^{N} V_j \phi_j, \tag{2.31a}$$

where $\mathbf{u}_h$ and $\mathbf{v}_h$ are the approximated solution and test function, respectively; ϕ_i and ϕ_j are the basis functions, typically chosen to have local support, meaning that they are non-zero only over a small number of elements; U_i and V_j are the unknown coefficients that need to be determined, and N is the total number of nodes in the mesh.

2.4.4 . Assembly of Global System of Equations

The solution and the test functions are approximated by a finite set of basis (or shape) functions. For each element e in the mesh, a local version of the weak form is computed, producing a local stiffness matrix $\mathbf{A}^e$ and a local force vector $\mathbf{F}^e$. These local matrices and vectors are derived from the integration of the weak form over each element, using the shape functions defined for that element.
To form the global system of equations, the contributions from each element need to be combined. For the entire mesh, this results in the following system:

$$\mathbf{A}\mathbf{U} = \mathbf{F}, \tag{2.32}$$

where $\mathbf{A}$ is the global stiffness matrix, $\mathbf{U}$ is the vector of unknown coefficients at each node in the mesh, and $\mathbf{F}$ is the global force vector. The global stiffness matrix $\mathbf{A}$ and the global force vector $\mathbf{F}$ are obtained by assembling the contributions from all the elements in the mesh:

$$\mathbf{A} = \sum_{e} \mathbf{A}^e \qquad \mathbf{F} = \sum_{e} \mathbf{F}^e \tag{2.33a}$$

The stiffness matrix $\mathbf{A}$ that results from this process is typically sparse. This means that most of the entries in the matrix are zero. The sparsity of $\mathbf{A}$ is one of the key advantages of FEM because it allows the system of equations to be solved efficiently, even for large-scale problems.

2.4.5 . Boundary conditions

Boundary conditions are essential in defining a well-posed problem in the FEM. They describe how the solution behaves at the boundaries of the computational domain and must be carefully incorporated into the weak form of the PDEs. Depending on the nature of the PDE and the problem being solved, different types of boundary conditions can be applied.

- Dirichlet Boundary Conditions: specify the value of the solution on the boundary of the domain. These conditions are used when the value of the variable being solved for (e.g., temperature, velocity, or displacement) is known or fixed at certain points on the boundary. In the context of FEM, Dirichlet boundary conditions are applied directly to the trial function u. This is typically done by modifying the basis functions to respect the boundary conditions. For example, if the solution u is fixed at a certain boundary node, then the corresponding entry in the global system of equations is adjusted to reflect this known value. Mathematically, Dirichlet boundary conditions take the form:

$$\mathbf{u} = \mathbf{g} \quad \text{on} \quad \partial\Omega_D \tag{2.34}$$

  where $\mathbf{g}$ is the prescribed value at the Dirichlet boundary $\partial\Omega_D$.

- Neumann Boundary Conditions: specify the value of the derivative of the solution normal to the boundary. These conditions are often used when the flux or gradient of a variable is known along the boundary. Mathematically, Neumann conditions are written as:

$$\frac{\partial \mathbf{u}}{\partial \mathbf{n}} = \mathbf{h} \quad \text{on} \quad \partial\Omega_N \tag{2.35}$$

  where $\frac{\partial \mathbf{u}}{\partial \mathbf{n}}$ is the derivative of the solution normal to the boundary $\partial\Omega_N$, and $\mathbf{h}$ is the prescribed flux or gradient. These conditions are incorporated into the weak form through the boundary integral term, which arises during the integration by parts process.

- Mixed Boundary Conditions: involve the application of both Dirichlet and Neumann conditions on different parts of the boundary. This is common in many real-world problems, such as fluid flow, where one part of the boundary may have a specified velocity (Dirichlet), while another part may have a specified stress or flux (Neumann). Incorporating mixed boundary conditions requires careful handling during the formulation and solution process, ensuring that both types of conditions are appropriately enforced in the weak form and the system of equations. The Dirichlet conditions directly modify the solution, while Neumann conditions are incorporated via the boundary integral terms.

2.4.6 . Numerical Solver

Depending on the nature of the underlying PDE, the resulting system can either be linear or nonlinear.

For linear systems (i.e., when the governing equations and boundary conditions are linear), solvers can be broadly classified into direct solvers and iterative solvers. Direct solvers, such as Gaussian elimination and LU decomposition, compute the solution by manipulating the matrix to eliminate unknowns. These methods are accurate but computationally expensive for large systems. Iterative solvers, such as Conjugate Gradient (CG) and GMRES, are more efficient for large, sparse systems. These methods iteratively approximate the solution by refining an initial guess, making them more suitable for the large-scale linear systems common in FEM.

For nonlinear systems of equations, the relationship between the solutions and the governing equations exhibits nonlinear behavior. These systems arise naturally in many fields, including problems governed by NSE. There are several iterative methods available for solving nonlinear systems. The Newton's method is the most widely used approach for nonlinear systems. It iteratively refines the solution by linearizing the system at each step using a Taylor series expansion. The Quasi-Newton methods approximate the Jacobian matrix, reducing the computational cost compared to Newton's method. Finally, the Newton-Krylov methods combine Newton's method with Krylov subspace solvers (like GMRES) to handle large systems without explicitly forming the Jacobian matrix.

Although these methods can be used in various contexts, Newton's method is the solver employed in this thesis due to its effectiveness for handling the nonlinearities characterizing the NSE and for its quadratic convergence when the initial guess is close to the true solution. Details of the numerical setting, including boundary conditions and time integration scheme, are provided in Chapter 5.

3 - Machine Learning (ML)

ML provides a vast range of tools and approaches, from traditional linear models to advanced deep learning techniques, each suited to different types of data and analytical goals. However, given the specific objectives of this thesis, the focus will be directed toward NNs and GNNs. On one hand, NNs provide a general structure for capturing complex relationships across data, while GNNs extend this capability specifically to graph-structured data, aligning with the scope of the present thesis.

3.1 . Basic structure of a Neural Network (NN)

At the core of any NN is the artificial neuron also known as *Perceptron* [Rosenblatt, 1958], which mimics the behavior of biological neurons in the brain. While biological neurons receive electrical impulses through synapses, artificial neurons receive input values. Although the biological analogy is inspiring, NNs are based on mathematical principles, where artificial neurons process inputs, apply transformations, and produce outputs that drive predictions or decisions.

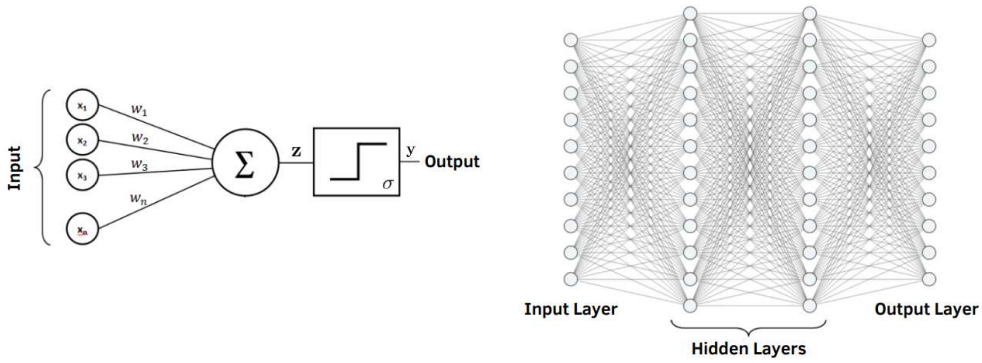


Figure 3.1: (Left) A Perceptron and (Right) a Multi-Layer Perceptron MLP representation.

Each neuron performs a simple mathematical operation by combining these inputs into a weighted sum, adding a bias term, and then passing the result through a non-linear function known as the activation function. With reference to Fig. 3.1 (Left), the operation of a Perceptron can be expressed mathematically as:

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{3.1}$$

where z is the *pre-activation* value, x_i are the inputs to the neurons, w_i are the weights applied to each input, b is the bias term and n is the number of input nodes. The weights w_i represent the strength of the connection between two neurons, while the bias b helps the neuron to adjust its output independently of its inputs, adding flexibility to the model. A compact representation of Eq. 3.1, particularly useful when handling inputs of higher dimensions, *i.e.* $\mathbf{x}_i \in \mathbb{R}^m$, can be expressed in matrix notation as:

$$\mathbf{z} = \mathbf{W}^T\mathbf{x} + \mathbf{b}, \tag{3.2}$$

where $\mathbf{x} \in \mathbb{R}^{n\times m}$ is the input tensor, $\mathbf{W}^T \in \mathbb{R}^{f\times n}$ is the weight tensor, $\mathbf{z} \in \mathbb{R}^{f\times m}$ is the *pre-activation* tensor, $\mathbf{b} \in \mathbb{R}^{f\times m}$ is the bias tensor, n is the number of input nodes, m is the dimension of $\mathbf{x}_i$ and f is the number of output nodes. Once the neuron computes the weighted sum $\mathbf{z}$, it passes the result through an activation function $\sigma(\mathbf{z})$, which introduces non-linearity and provides the output $\mathbf{y}$

$$\mathbf{y} = \sigma(\mathbf{z}). \tag{3.3}$$

Non-linearity is essential because, without it, the NN would behave like a simple linear model, unable to capture complex patterns in data. Some commonly used activation functions include (non-exhaustive list):

- Linear Activation function: is the simplest form of activation, where the output is directly proportional to the input. However, since it does not introduce any non-linearity, a network with only linear activations behaves like a linear model regardless of the number of layers:

$$\sigma(\mathbf{z}) = \mathbf{z}. \tag{3.4}$$

- Sigmoid function: maps the output to a range between 0 and 1, and it's useful in the output layer for binary classification, where the output represents a probability.:

$$\sigma(\mathbf{z}) = \frac{1}{1 + e^{-\mathbf{z}}}. \tag{3.5}$$

- Hyperbolic Tangent (Tanh): similar to the Sigmoid function but outputs values between -1 and 1, often leading to faster convergence in certain tasks:

$$\sigma(\mathbf{z}) = \tanh(\mathbf{z}) = \frac{e^{\mathbf{z}} - e^{-\mathbf{z}}}{e^{\mathbf{z}} + e^{-\mathbf{z}}}. \tag{3.6}$$

- Rectified Linear Unit (ReLU): the most popular activation function in deep learning, ReLU outputs $\mathbf{z}$ if $\mathbf{z} > 0$ and 0 otherwise. ReLU is computationally efficient and well-suited for hidden layers in deep neural networks:

$$\sigma(\mathbf{z}) = \max(0, \mathbf{z}) \tag{3.7}$$

- Softmax: is used in the output layer of neural networks when handling multi-class classification problems. It converts the raw output scores into probabilities that sum to 1, providing a normalized distribution across all possible classes. In the following equation, $\mathbf{z}_i$ represents the score for class i, and the denominator normalizes the values across all classes j:

$$\sigma(\mathbf{z}_i) = \frac{e^{\mathbf{z}_i}}{\sum_j e^{\mathbf{z}_j}}. \tag{3.8}$$

The choice of activation function can significantly affect the performance of a NN. There is no universal rule for selecting the best activation function; instead, the optimal choice heavily depends on the specific problem being addressed, and empirical experimentation is often necessary. The artificial neuron thus serves as the building block for all NNs, where the weights and biases are adjustable parameters that are learned during the training process.
When Perceptrons start to be stacked together, a more complex network emerges, known as *Multi-Layer Perceptron*, see Fig. 3.1 (Right) [Rosenblatt, 1958]. The MLP extends the concept of the Perceptron by introducing multiple layers of neurons, organized into an input layer, one or more hidden layers, and an output layer. Each layer can be fully connected to the next, meaning that each neuron in a given layer is connected to every neuron in the following layer.

3.2 . Functioning of a Neural Network (NN)

The functioning of a NN, particularly during training, revolves around two key processes: *forward propagation* and *backward propagation* [Rumelhart et al., 1986]. These two steps work together to enable the network to learn from data and improve its predictions.
With reference to Fig. 3.1 (Right), in the forward propagation step, input data is fed into the network, processed through each hidden layer of neurons, and transformed into an output. Starting from the input layer, the flowing of information in an MLP can be mathematically described as follows, in matrix notation

- Input to the first hidden layer:

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \tag{3.9a}$$
$$\mathbf{a}^{(1)} = \sigma(\mathbf{z}^{(1)}). \tag{3.9b}$$

- Subsequent hidden layers:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \tag{3.10a}$$
$$\mathbf{a}^{(l)} = \sigma(\mathbf{z}^{(l)}). \tag{3.10b}$$

- Output layer:

$$\mathbf{y} = \mathbf{W}^{(L)}\mathbf{a}^{(L-1)} + \mathbf{b}^{(L)}, \tag{3.11}$$

where $\mathbf{W}^{(i)}$ are the weight matrices, $\mathbf{x}$ the input vector, $\mathbf{y}$ the output vector, $\mathbf{a}^{(i)}$ the inner layer vectors, $\mathbf{b}^{(i)}$ the bias vectors, σ the non-linear activation functions; the apex represents the step index in the NN structure, ranging from 1 (input layer) to L (output layer). The final output of the network is the result of this process and represents the network's prediction. Depending on the task at hand, the output can be interpreted as a classification, regression value, or any other target.
Once the forward pass is complete, the network compares its predicted output $\hat{\mathbf{y}}$ with the actual target value $\mathbf{y}$ using a loss function. The loss function quantifies the error between the predicted and actual values, guiding the network in learning how to improve its predictions. Among several loss functions available, the most common include (non-exhaustive list):

- Mean Squared Error (MSE): for regression tasks. It calculates the average squared difference between the predicted and actual values as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2. \tag{3.12}$$

- Cross-Entropy Loss: for classification tasks. It compares the predicted probability distribution $\hat{\mathbf{y}}$ with the true distribution $\mathbf{y}$, driving the model to assign higher probabilities to the correct classes:

$$\text{Cross-Entropy} = -\sum_{i=1}^{n}\mathbf{y}_i \log(\hat{\mathbf{y}}_i). \tag{3.13}$$

After the loss is calculated, the network enters the backward propagation step, where it adjusts its weights and biases to reduce the loss in future predictions. Backward propagation works by computing the gradients of the loss function with respect to each weight and bias in the network using the chain rule from calculus. This process estimates how much each weight and bias in the network contribute to the error. Then, weights and biases of the NN are adjusted using an optimization algorithm. Among several optimization methods available, the most common include (non-exhaustive list):

- Stochastic Gradient Descent (SGD): SGD is one of the most commonly used optimization algorithm. The term *stochastic* refers to the randomness introduced by using only a random subset $\mathbf{x}_N \subset \mathbf{X}$ of data extracted from the dataset $\mathbf{X}$, rather than the full dataset, when computing the required gradients. This randomness often helps escape local minima or saddle points, allowing the network to find a better global

optimum.
Each update of the weights for the $l\text{-}th$ layer of the NN in SGD is calculated as:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \frac{\partial \mathcal{L}(\mathbf{x}_N)}{\partial \mathbf{W}^{(l)}} \tag{3.14a}$$

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \eta \frac{\partial \mathcal{L}(\mathbf{x}_N)}{\partial \mathbf{b}^{(l)}} \tag{3.14b}$$

where $\mathbf{x}_N$ represents the portion of the data used to compute the gradients and η is the *learning rate* which controls the step size in the gradient direction. $\partial \mathcal{L}(\mathbf{x}_N)/\partial \mathbf{W}^{(l)}$ and $\partial \mathcal{L}(\mathbf{x}_N)/\partial \mathbf{b}^{(l)}$ are the gradients of the loss function with respect to the weights matrix and the bias vector, respectively.

- Adaptive Moment Estimation (ADAM): ADAM is an advanced optimization algorithm. Instead of using a single global learning rate, ADAM maintains adaptive learning rates for each parameter based on the first and second moments (mean and uncentered variance) of the gradients.

A complete forward-backward loop on each data $\mathbf{x}$ in the training dataset $\mathbf{X}$ is called an *epoch*. The number of epochs determines how many times the entire dataset is passed through the network during the training phase. The training process continues until the loss converges, meaning that additional training no longer leads to significant improvements, or until a specified number of epochs is reached. The network typically improves over multiple epochs, but it's important to monitor the loss to avoid some of the common problems found when training a NN.

3.3 . Neural Networks (NNs) as Universal Approximator

One of the most remarkable properties of NNs is their ability to function as *Universal Approximators*. This concept, formalized by Cybenko [1989] for sigmoid activation functions and later extended by Hornik [1991], is known as the Universal Approximation Theorem. It's definition asserts:

Universal Approximation Theorem. *Let σ be a continuous, bounded, and non-constant activation function. Then for any continuous function f defined on a compact subset $K \subset \mathbb{R}^n$ and for any $\epsilon > 0$, there exists a feedforward neural network with a single hidden layer and a finite number of neurons such that the network's output $\hat{f}$ approximates f within ϵ, i.e.,*

$$|f(\mathbf{x}) - \hat{f}(\mathbf{x})| < \epsilon \quad \forall \mathbf{x} \in K$$

In other words, this theorem states that a feedforward NN, with at least one hidden layer containing a sufficient number of neurons, can approximate

any continuous function to any desired degree of accuracy. The theorem highlights the immense expressive power of NNs, making them capable of representing highly complex, non-linear relationships between inputs and outputs. The two conditions posed in the theorem definition are critical requirements; firstly, the activation function must be non-linear. In the case of a linear activation function, the network's output would simply be a linear combination of the inputs, limiting the network to modeling only linearly separable data. Secondly, the activation function must be bounded, meaning that there exists some constant $M > 0$ such that $|f(\mathbf{z})| \leq M$ for all $\mathbf{z} \in \mathbb{R}^n$.
The Universal Approximation Theorem guarantees that NNs can, in principle, learn and approximate any continuous function as long as they have sufficient capacity, meaning enough neurons and layers.
However, it's important to note that the theorem only guarantees the existence of a network capable of approximating the function, but it doesn't provide practical guidance on how to design such a network or how to efficiently train it. In practice, achieving good approximations often requires careful architecture design, hyperparameter tuning, and a well-chosen training algorithm.
In addition, the theorem doesn't account for the generalization ability of the network. A model that perfectly approximates a function on the training data may not generalize well to unseen data if it has been overfitted. Thus, while neural networks have the power to approximate any function, considerations such as model expressivity, complexity, and generalization are crucial for real-world applications.

3.4 . Types of Neural Networks (NNs)

NNs come in a great variety of architectures, each suited to different types of data and tasks. The architecture chosen for a specific task depends on the nature of the data, the complexity of the problem, and the desired output.
Below are the primary types of neural networks, their structural characteristics, and their typical applications (non-exhaustive list):

- Multi-Layer Perceptron (MLP)s: the simplest form of Neural Network, characterized by the presence of one or more hidden layers between those of input and output. In these networks, data flows in a unidirectional manner, passing through the various hidden layers, with each neuron in one layer connected to all neurons in the subsequent layer. MLP utilize non-linear activation functions (Sec. 3.1) which allow them to learn complex representations from the data. This architecture is particularly effective for classification and regression tasks, making them suitable for a wide range of applications, including image recognition, data analysis, and predictive modeling.

- Convolutional Neural Network (CNN)s: specifically designed to process data with a grid-like topology, such as images or videos. CNNs utilize a specialized layer known as the convolutional layer, which applies filters or kernels over the input data to automatically detect and learn important features such as edges, textures, or complex patterns. This is followed by pooling layers that reduce the spatial dimensions of the data, helping to reduce the computational cost by retaining only the most important features.

- Recurrant Neural Networks (RNN)s: designed to handle sequential data, where the order of the inputs is important. RNNs have connections that form cycles, allowing the network to maintain a memory of previous inputs by passing information forward through time. This makes RNNs especially suited for tasks that require the network to reproduce the temporal correlation and dynamics underneath the data.

- Autoencoders: primarily used for unsupervised learning. The architecture consists of two main components: an encoder, which compresses the input data into a low-dimensional latent space, and a decoder, which reconstructs the original input from this compressed representation. The goal is to learn an efficient representation of the data in the latent space that captures the most salient features. Autoencoders are widely used for dimensionality reduction, anomaly detection, and data generation.

- Generative Adversial Network (GAN)s: consist of two competing NNs: a generator and a discriminator. The generator's task is to create realistic-looking fake data samples (e.g., images, videos, or audio), while the discriminator tries to distinguish between real and fake data samples. The two networks are trained simultaneously in a game-theoretic framework, where the generator improves by "fooling" the discriminator, and the discriminator improves by getting better at detecting fakes. GANs have revolutionized the field of generative models, producing highly realistic images, video, and audio content.

- Graph Neural Network (GNN)s: designed to handle data represented in graph structures, where relationships between entities are defined by edges. GNNs excel at capturing complex dependencies and relationships between entities in unstructured data. In GNNs, nodes represent data points, and edges represent the relationships or interactions between them. Due to their unique ability to model relationships between entities, GNNs are increasingly used in fields like network analysis, social network modeling, recommendation systems, and molecular chemistry. More recently, GNNs have been applied to CFD to model fluid

interactions, making them an emerging tool in this field.

3.5 . Limitations of Neural Networks (NNs)

Despite their success, traditional NNs face several limitations that restrict their scalability, interpretability, and efficiency in real-world applications.
A first concern is about the scalability of these structures. As NNs grow in size to handle complex tasks, their computational complexity increases exponentially. Training deep networks with millions (or billions) of parameters requires substantial computational resources and can take significant time, especially when dealing with large datasets. This high resource demand can make NNs impractical for real-time or resource-constrained environments. Directly related to this problem is the high energy consumption and, as extension, the environmental impact concerns, emphasizing the need for more efficient and sustainable AI technologies.
Another key limitation of NNs is their black box nature, which results from their lack of transparency in decision-making. The internal processes of a NN are indeed not easily interpretable, making some outputs difficult to explain. This characteristic can pose challenges in sensitive domains such as healthcare, finance, and autonomous systems, where decisions need to be both explainable and reliable.
Moreover, NNs are data-hungry and require large amounts of labeled data to perform well. This dependence can be a bottleneck in data-scarce fields. Additionally, traditional NNs often struggle with generalization to unseen data, especially when trained on narrow or biased datasets. NNs are often inefficient in terms of parameter utilization. While they are powerful universal function approximators, traditional feedforward architectures can require a disproportionately large number of parameters to represent even simple functions in low-dimensional spaces. In an attempt to overcome these limitations, researchers are developing advanced NN architectures and research focuses on energy-efficient models, enhancing explainability through Explainable AI [Barredo Arrieta et al., 2020], and improving generalization using techniques like transfer learning [Thrun and Pratt, 1998].

3.6 . Graph Theory

Graph theory is a fundamental branch of mathematics that deals with studying the properties and relationships of graphs [Wilson, 1996]. Graphs are mathematical structures used to represent pairwise connections between objects. These structures are pivotal in a wide range of fields, including computer science, physics, engineering, biology, and social sciences. In the context of this thesis, graph theory provides the foundational framework necessary to

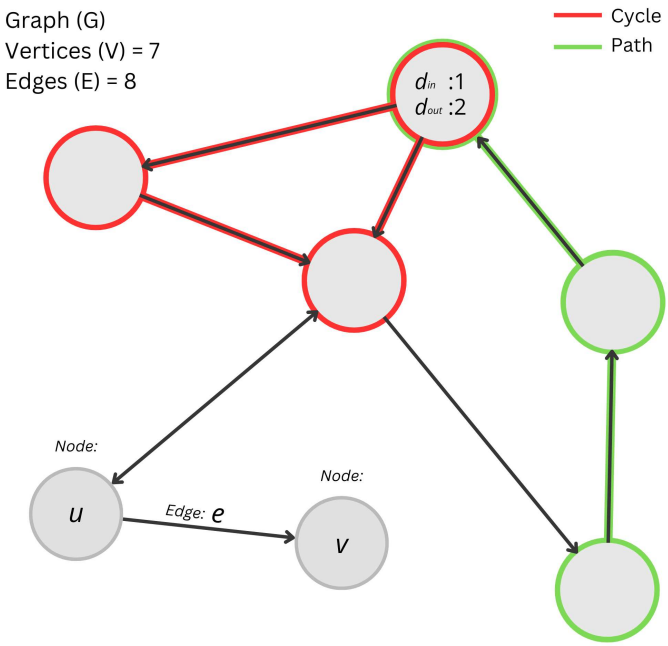


Figure 3.2: A pictorial representation of a graph G (Eq. 3.15)

understand the working principles of GNNs, the central model explored in this work.

3.6.1 . Introduction to Graph Theory

Formally, a graph $\mathbf{G}$, whose sketch can be seen in Fig. 3.2, is defined as an ordered pair

$$\mathbf{G} = (\mathbf{V}, \mathbf{E}), \tag{3.15}$$

where $\mathbf{V}$ represents the set of vertices (or nodes) and $\mathbf{E}$ is a set of edges connecting pairs of vertices. The vertices $\mathbf{V}$ can represent entities or objects in a system, while the edges $\mathbf{E}$ define the relationships or interactions between these entities.
Graphs can be categorized based on their structural characteristics and the nature of connections between their vertices:

- Directed and Undirected Graphs: An Undirected graph is one in which edges have no direction, meaning that an edge between two vertices u and v, denoted as (u, v) is identical to (v, u). Undirected graphs are used in situations where relationships are mutual, such as friendships in social networks. On the other hand, when (u, v) is not the same as (v, u) and the edge represents a one-way relationship, such as the flow of information in communication networks or dependencies in task scheduling, the graph is called directed graph.

- Weighted and Unweighted Graphs: A weighted graph assigns a numerical weight to each edge, representing the strength, cost, or capacity of the relationship between the connected vertices. For example, in a transportation network, weights could represent distances or travel times between cities. An unweighted graph treats all edges as equal, with no specific value assigned to the relationships.

- Simple Graphs and Multigraphs: A simple graph contains no loops (edges that connect a vertex to itself) and no multiple edges between the same pair of vertices. This is the most basic type of graph and is widely used in many applications where redundant or reflexive relationships are not meaningful. A multigraph, by contrast, allows multiple edges between the same pair of vertices, which can represent multiple types or instances of relationships.

- Complete Graphs: A complete graph is one in which every pair of distinct vertices is connected by a unique edge. These graphs are often used in theoretical contexts, as they represent the maximum number of relationships between a set of nodes. In practical applications, they are less common due to their density and complexity.

- Bipartite Graphs: A bipartite graph is one where the vertex set $\mathbf{V}$ can be divided into two disjoint subsets $\mathbf{V}_1$ and $\mathbf{V}_2$, such that all edges connect a vertex in one set to a vertex in another set. There are no edges between vertices within the disjoint sets.

Depending on the types of connected information upon which the graph is built, mathematical differences there exists and, as a consequence, different types of graphs shows different properties and are suited for different analysis and applications.

3.6.2 . Mathematical representation of Graphs

Graphs can be mathematically represented in multiple ways depending on the type of analysis or application. Each representation provides a different perspective on the properties of the graph's structure and connectivity. This section explores the most common ways to represent a graph mathematically, as well as key properties that provide deeper insights into its structure.

- Adjacency Matrix: one of the most widely used representations of a graph. Given a graph $\mathbf{G}$ (Eq. 3.15) with n vertices, the adjacency matrix $\mathbf{A}$ is an $n \times n$ matrix that encodes the presence or absence of edges between pairs of vertices. The matrix element A_{ij} is defined as:

$$A_{ij} = \begin{cases} 1 & \text{if there is an edge between vertex } v_i \text{ and } v_j, \\ 0 & \text{otherwise.} \end{cases} \tag{3.16}$$

For weighted graphs, the adjacency matrix generalizes to store the weight of the edge between two vertices:

$$A_{ij} = \begin{cases} w_{ij} & \text{if there is an edge between vertex } v_i \text{ and } v_j, \\ 0 & \text{otherwise.} \end{cases} \tag{3.17}$$

For undirected graphs, the matrix is symmetric (i.e., $A_{ij} = A_{ji}$), while for directed graphs, the matrix is not necessarily symmetric, reflecting the directionality of the edges.

- Incidence Matrix: captures the relationship between vertices and edges. Let $\mathbf{G}$ have n vertices and m edges. The incidence matrix $\mathbf{B}$ is an $n \times m$ matrix, where each row corresponds to a vertex and each column corresponds to an edge. The matrix element B_{ij} is defined as:

$$B_{ij} = \begin{cases} 1 & \text{if vertex } v_i \text{ is incident to edge } e_j, \\ 0 & \text{otherwise.} \end{cases} \tag{3.18}$$

Alongside their representation, many properties can be defined for a graph.

- Degree of a Vertex: It represents the number of edges connected to a given vertex. In undirected graphs, the degree of a vertex v_i denoted as $d(v_i)$, is simply the number of edges that involve v_i. In directed graphs, an in-degree and an out-degree can be defined on each vertex. Mathematically, they are defines as:

$$d_{in}(v_i) = \sum_{j=1}^{n} \mathbf{A}_{ji}, \qquad d_{out}(v_i) = \sum_{j=1}^{n} \mathbf{A}_{ij}. \tag{3.19a}$$

  They indicate respectively the number of incoming and outgoing edges for the vertex v_i

- Path and Cycle: A path in a graph is a sequence of edges that connects a sequence of distinct vertices. Formally, a path P of length k is a sequence of vertices $(v_0, v_1, \ldots, v_k)$ such that there exists an edge between v_i and v_{i+1} for all $0 \leq i < k$. Paths are fundamental for traversing graphs and are used in algorithms such as shortest path algorithms.
  A cycle is a special type of path where the starting and ending vertices are the same. Formally, a cycle C is a path $P = (v_0, v_1, \ldots, v_k)$ such that $v_0 = v_k$. These two concepts are visually represented in Fig. 3.2.

- Graph Diameter and Radius: The diameter of a graph is the longest among the shortest path between any two vertices. Formally, if $d(u, v)$ represents the shortest path between vertices u and v, the diameter of the graph $\mathbf{G}$ is given by:

$$\text{diam}(\mathbf{G}) = \max_{u,v \in V} d(u, v). \tag{3.20}$$

The diameter gives an indication of the spread of the graph, representing the maximum distance between any two vertices. The radius of a graph is the minimum eccentricity of any vertex. The eccentricity $e(v)$ of a vertex v is the greatest distance from v to any other vertex. The radius is defined as:

$$\text{radius}(\mathbf{G}) = \min_{v \in V} e(v). \tag{3.21}$$

This measure provides insight into the centrality of the most central vertex in the graph.

- Centrality and Importance of Vertices: The centrality of a vertex quantifies its relative importance within a graph. One simple measure is the degree of centrality, which is proportional to the number of edges connected to a vertex. Vertices with high degree of centrality are often seen as influential or central to the graph's structure. Other centrality measures, such as betweenness centrality and closeness centrality, focus on the role of a vertex in facilitating connections between other vertices or its proximity to other vertices within the network.

3.7 . Graph Neural Network (GNN)

GNNs were introduced to extend traditional NNs to graph-structured data, addressing the limitations of standard models designed for grid-like structures (e.g., images, sequences). The foundational work by Scarselli et al. [2008] formalized GNNs, focusing on the need to model complex relationships in fields such as social networks and molecular chemistry. Early GNNs aimed to process data with irregular structures, where nodes and edges represent entities and their relationships.
The key mechanism in GNNs is *message passing* (Sec. 3.7.1), where each node aggregates information from its neighbors, updating its representation. Through multiple layers, the network learns complex relationships by iteratively exchanging information across the graph. This allows GNNs to capture both local and global structural patterns, making them powerful for relational data modeling.

3.7.1 . Core principles of GNNs

GNNs are a powerful extension of traditional NNs, designed to process graph-structured data. Unlike data arranged in regular grids (e.g., images or sequences), graphs consist of nodes (vertices) and edges, representing complex relationships between entities. The foundational computational framework of GNNs is known as *message passing* or *neighborhood aggregation*, which enables nodes to exchange information iteratively with their neighbors, updating their representations layer by layer.

This core operation can be expressed using the following general message passing formula:

$$\mathbf{x}_i^{(k+1)} = \gamma^{(k)} \left(\mathbf{x}_i^{(k)}, \bigoplus_{j \in \mathcal{N}(i)} \phi^{(k)}(\mathbf{x}_i^{(k)}, \mathbf{x}_j^{(k)}, \mathbf{e}_{ij}) \right) \tag{3.22}$$

This formula encapsulates the entire message passing mechanism, and understanding it is crucial to grasp how GNNs operate. Each component of this equation is examined in the following, along with the theoretical principles that govern them.

- Node Features $\mathbf{x}_i^{(k)}$: The feature vector $\mathbf{x}_i^{(k)}$ represents the state of the node i at layer k of the GNN. Initially, these features may represent intrinsic properties of the node (e.g., molecular structure in a chemistry graph or user profile in a social network). As the MP iterates, this feature vector is updated through interactions with neighboring nodes, progressively incorporating more information from the graph structure.
- Neighborhood $\mathcal{N}(i)$: $\mathcal{N}(i)$ denotes the set of neighbors of node i. GNNs are local aggregators, meaning that at each step of the MP, a node only interacts with its immediate neighbors. As the GNN goes deeper (i.e., through more MP steps), the node indirectly aggregates information from nodes further away in the graph, eventually capturing global graph patterns.
- Message Function $\phi^{(k)}$: The function $\phi^{(k)}$ computes the message passed from each neighboring node j to node i. This function depends on the features of both the source node j and the target node i, as well as any additional edge features $\mathbf{e}_{ij}$ that may exist between them. The message function ϕ is typically a learnable transformation, such as an MLP. The purpose of this function is to encode the interaction between neighboring nodes in a way that is useful for the task at hand (e.g., node classification, link prediction).
- Aggregation Function $\bigoplus$: The symbol $\bigoplus$ represents the aggregation operation, which combines the messages $m_{i,j}$ received from all neighbors $j \in \mathcal{N}(i)$. Common choices for this aggregation include:
  - Summation: which aggregates the messages by adding them together:
  $$\sum_{j \in \mathcal{N}(i)} m_{i,j}. \tag{3.23}$$
  - Mean: which takes the average of the messages:
  $$\frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} m_{i,j}. \tag{3.24}$$

- Maximization: which selects the maximum message value across all neighbors:

$$\max_{j \in \mathcal{N}(i)} m_{i,j}. \tag{3.25}$$

The choice of aggregation function can significantly impact the performance of the GNN and heavily depends on the task at hand.

- Update Function $\gamma^{(k)}$: After aggregating the messages from neighbors, the update function $\gamma^{(k)}$ combines the aggregated message with the current feature vector of the node to produce the updated node representation $\mathbf{x}_i^{(k+1)}$. This step allows the node to incorporate information from its neighbors while retaining aspects of its previous state. The update function is often implemented as a learnable neural network layer, such as an MLP.

The procedure of the MP is repeated for multiple layers (or iterations), allowing information to propagate across the graph. At each layer k, a node aggregates information from its neighbors, updates its feature vector, and passes this information to the next layer. After K layers, each node has aggregated information from nodes up to K graph steps away, thus capturing broader graph structures. This iterative nature is what allows GNNs to model complex dependencies in the graph. A shallow GNN (e.g., with one or two layers) may only capture local structures, while a deeper GNN can learn more global patterns by aggregating information from nodes that are far apart in the graph. However, deeper GNNs can also suffer from issues like over-smoothing, where node features become indistinguishable as more layers are added.

3.8 . A custom Graph Neural Network (GNN) architecture

Building upon the general principles of GNNs introduced in Sec. 3.7.1, this section presents the custom GNN architecture specifically designed to tackle challenges in CFD. The architecture is tailored to fit the physics of the fluid system, characterized by convective and diffusive dynamics, the flow properties like the Reynolds number and the chosen numerical approximation defined on an unstructured mesh.
This custom GNN model will be used throughout the present thesis as the primary architecture for all the proposed CFD applications. It is specifically designed to interact with unstructured CFD data and to adapt across a wide range of geometries and problem settings, making it a versatile solution for the problems tackled in this work. MP, as referred to our custom GNN, to be thought as node centered, involves three fundamental steps:

1. Message Creation: Each node i initiates an embedding state represented by an array $\mathbf{h}_i$. Initially set to zero, this vector accumulates and handles information as the MP proceeds. The dimension d_h of $\mathbf{h}_i$ is constant across all nodes and is a key model hyperparameter. Note that the embedded state itself does not have a direct physical interpretation.

2. Message Propagation: Information is then propagated between nodes. To capture the convective and diffusive dynamics of the underlying CFD systems, messages are transmitted bidirectionally between connected nodes. Given a generic pair of connected nodes i and j and a directed connection between them $\mathbf{a}_{ij}$ from i to j, the abstract information (or message) generated on them is defined as:

$$\boldsymbol{\phi}_{i,j}^{(k)} = \zeta^{(k)}(\mathbf{h}_i^{(k-1)}, \mathbf{a}_{ij}, \mathbf{h}_j^{(k-1)}), \tag{3.26}$$

where $\mathbf{h}_i^{(k-1)}$ is the embedded state from the previous MP layer $k-1$, and $\zeta^{(k)}$ is a differentiable operator, such as, in our case, an MLP. Note that swapping the indices i and j in Eq. 3.26, gives the definition for the message that flows from j to i. Depending on the number of j connected nodes in the neighboring set of i, namely $\mathcal{N}_i$, for each node i the global outgoing message is then computed as:

$$\boldsymbol{\phi}_{i,\rightarrow} = \bigoplus_{j \in \mathcal{N}_i} \boldsymbol{\phi}_{i,j} \tag{3.27}$$

where $\bigoplus$ is an arbitrary differentiable, permutation invariant function, e.g., sum, mean or max.

3. Message Aggregation: Each node i aggregates the collected information to update its embedded state $\mathbf{h}_i^{(k)}$:

$$\mathbf{h}_i^{(k)} = \mathbf{h}_i^{(k-1)} + \alpha \Psi^{(k)}(\mathbf{h}_i^{(k-1)}, \mathbf{G}_i, \boldsymbol{\phi}_{i,\rightarrow}^{(k)}, \boldsymbol{\phi}_{i,\leftarrow}^{(k)}, \boldsymbol{\phi}_{i,\circlearrowleft}^{(k)}), \tag{3.28}$$

where $\mathbf{G}_i$ represents the external injected quantities, *i.e.* the data input to the GNN. This input, provided at each update k, depends on the specific application, and it will be clarified for each of them. The vectors $\boldsymbol{\phi}_{i,\rightarrow}^{(k)}$ and $\boldsymbol{\phi}_{i,\leftarrow}^{(k)}$ represent respectively the message sent to and received from all the neighboring nodes. The vector $\boldsymbol{\phi}_{i,\circlearrowleft}^{(k)}$ is the self-message that the node i send to itself in order to maintain the node's own information while aggregating messages from its neighbors. Their mathematical definition, with the appropriate change in notation, is expressed in Eq. 3.26. The term $\Psi^{(k)}$ is a differentiable operator, typically an MLP, used to handle together the gathered information. The term α is a hyperparameter relaxation coefficient controlling the update scale.

By the end of the message passing process, each node's embedded state has been updated k times, integrating information from other nodes in the graph. Research has shown that the choice of this hyperparameter k is crucial: it should be adapted to the specific mesh to improve the network's generalization capability across graphs of varying sizes. The study by Nastorg [2024] demonstrates that k should ideally be proportional to the graph's diameter (Sec. 3.6.2), ensuring effective information propagation throughout the entire structure. [Donon et al., 2020], suggest that k should match or exceed the diameter of the graph. However, since the graphs in the dataset used in this study exhibit a relatively consistent diameter, we opted to optimize this hyperparameter using genetic algorithms (Sec. 3.8.3). This approach allows the network to efficiently adapt to the dataset's structure, without overextending beyond the scope of this thesis, as hyperparameter tuning is not our primary focus. Interestingly, Nastorg [2024] explored a recurrent or adaptive architecture that could theoretically allow for variable k, dynamically adjusting to the graph's structure.
At the end of the MP process, the latest k-updated embedded state on each node i is projected back into a physical state as prediction of the required target, which depends on the specific learning task. A differentiable operator such as an MLP, namely a Decoder D, is tasked with this latter operation.
It is important to highlight the custom GNN architecture presented here diverges from standard GNN models (Sec. 3.7) as it address specific requirements that are typical of CFD applications. First, external quantities G are introduced at each update step, allowing the network to incorporate domain-specific information. Secondly, the MP process is bidirectional: it considers both incoming and outgoing information at each node, a design choice that reflects the conservation principles and convective-diffusive dynamics of the governing physical laws. Finally, as the aggregation function, the mean is chosen to maintain permutation invariance with respect to the number of neighboring nodes. This decision is critical for working with unstructured meshes, where the neighborhood size of each node may vary across cases and nodes, allowing the model to generalize effectively over different mesh configurations.

3.8.1 . Data Structuring

Applying GNNs to unstructured data requires their graph representation. In order to obtain the CFD-GNN interface, each mesh node is aligned with a GNN node. To this end, the CFD data are structured into tensors that maintain adjacency properties from the mesh. Specifically, for each case in the ground truth dataset, different data structure are generated:

- A matrix $\mathbf{A} \in \mathbb{R}^{n_i \times d_h}$, where n_i is the number of nodes in the mesh and d_h is the dimension of the embedded state defined on each node.

$\mathbf{A}$, therefore, is a tensor stacking together all the embedded arrays h_i defined on all the nodes.

- A matrix $\mathbf{C} \in \mathbb{R}^{c \times 2}$, where c is the number of mesh edges, defining the nodes connections. $\mathbf{C}$, therefore, is a tensorial representation of the adjacency scheme of the mesh.
- A matrix $\mathbf{D} \in \mathbb{R}^{c \times 2}$, containing the distances between connected nodes in the x and y directions. $\mathbf{D}$, therefore, express the properties, in the meaning of nodes distances, of the adjacency scheme of the mesh.

$$\mathbf{A}_{n_i,d_h} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,d_h} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,d_h} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n_i,1} & a_{n_i,2} & \cdots & a_{n_i,d_h} \end{bmatrix},$$

$$\mathbf{C}_{c,2} = \begin{bmatrix} i_1 & j_1 \\ i_2 & j_2 \\ \vdots & \vdots \\ i_c & j_c \end{bmatrix}, \qquad \mathbf{D}_{c,2} = \begin{bmatrix} x_{i_1} - x_{j_1} & y_{i_1} - y_{j_1} \\ x_{i_2} - x_{j_2} & y_{i_2} - y_{j_2} \\ \vdots & \vdots \\ x_{i_c} - x_{j_c} & y_{i_c} - y_{j_c} \end{bmatrix}.$$

Each column of $\mathbf{A}$ serves as a feature vector for neurons in the MLPs used in the GNN (ζ, Ψ, and the decoder D). The structure of these MLPs is instead defined by the dimension d_h of the embedded state, while the number of nodes n_i corresponds to the feature count per neuron. This setup allows us to apply the same MLPs architectures across different CFD simulations, regardless of the geometry or node count, as the number of nodes does not affect the underlying structure of the MLPs. This approach makes the presented custom GNN particularly well-suited for interacting with unstructured meshes, learning from various geometries and configurations.

3.8.2 . GNN Training Algorithm

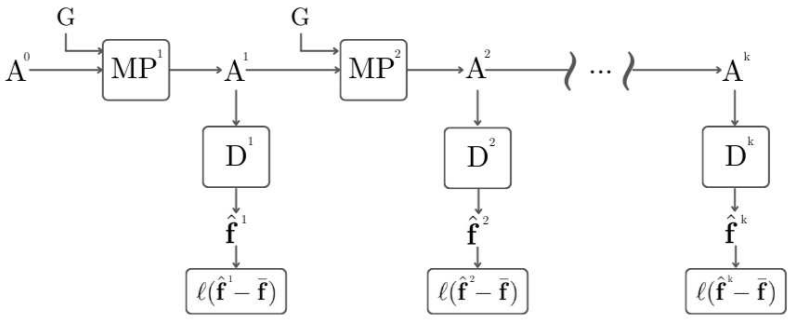


Figure 3.3: The overall framework of our GNN training process. MP^k are the message passing algorithms; D^k are the k decoders trainable MLPs; $\mathbf{A}^k$ are the k matrices containing the embedded states from each node; $\mathbf{G}$ is the vector containing the input injected in the GNN. Figure inspired by Donon et al. [2020].

The training framework for the GNN is illustrated in Fig. 3.3. The process begins with $\mathbf{A}^0$, a matrix of zero-initialized embedded states. This matrix, along with external inputs $\mathbf{G}$, is provided to the first message passing algorithm MP^1. The updated embedded state matrix $\mathbf{A}^1$, then, passes through a decoder D^1, an MLP tasked with reconstructing the target physical state g. The predicted target $\hat{g}^1$ is compared with the ground truth corresponding data g using a loss function:

$$\ell^k = \frac{1}{n_i} \sum_{i=1}^{n_i} (g_i^k - \hat{g}_i)^2 \tag{3.29}$$

where n_i is the number of nodes and k indicates the layer number of the MP process. This process is then repeated across the k layers of the GNN. Following the intuition of Donon et al. [2020], all these intermediate loss values from the different update layers are considered in a global loss function L, in order to robustify the learning process:

$$L = \sum_{k=1}^{\bar{k}} \gamma^{\bar{k}-k} \cdot \ell^k \tag{3.30}$$

where, $\bar{k}$ is the number of update layers, and γ is a hyperparameter controlling the weight of each of them. As the MP process goes on, each node collects more and more information. The exponential term $\gamma^{\bar{k}-k}$ ensures that later updates, which are supposed to be richer in information, have greater influence on the learning process.

3.8.3 . GNN hyperparameters

The structure and performance of a neural network, are heavily influenced by its *hyperparameters*. These hyperparameters are values set before the training process begins and are not learnable from the data during training, but instead, they govern the capacity and behavior of the model. We can categorize hyperparameters into two main types: *model hyperparameters* and *process hyperparameters*.

- Model hyperparameters: These define the capacity of the NN, which refers to the model's ability to approximate complex, non-linear functions. The capacity is directly influenced by factors such as the number of layers, the number of neurons in each layer, and the dimensionality of the embedded node features. Higher capacity allows the model to represent more complex relationships in the data but comes at the cost of increased computational demand. In our GNN, model hyperparameters determine how well the network can capture the patterns of fluid dynamics represented in the mesh-based CFD data.

- Process hyperparameters: These govern the training process itself, affecting the duration, computational costs, and efficiency of the training phase. Process hyperparameters include aspects such as the learning rate, which controls how fast the model's weights are updated, and the regularization techniques applied to prevent overfitting. Proper tuning of these hyperparameters is essential to ensure that the model converges effectively while minimizing training time and computational resource usage.

Given the need for a computationally efficient yet expressive model, the hyperparameters defining the NNs architecture must be optimized carefully. Unlike continuous parameters that can be fine-tuned using gradient-based methods, many of the hyperparameters involved in NN design, such as the number of layers or neurons, are discrete and cannot be adjusted using standard gradient-based optimization techniques. For this reason, gradient-free optimization algorithms are used to explore the hyperparameter space efficiently.
Various libraries and tools are available for automating the hyperparameter optimization process by systematically searching through combinations of possible hyperparameters and pruning unpromising configurations. In this work, we employed the open-source optimization library `Optuna` [Akiba et al., 2019], which combines efficient searching with advanced pruning strategies to identify the best-performing set of hyperparameters based on validation metrics.
`Optuna` operates by first defining the search space, which includes the possible ranges or values for each hyperparameter. It then evaluates the GNN's

performance over multiple trials, each corresponding to a different set of hyperparameters. During this process, `Optuna` applies dynamic pruning, where trials that are unlikely to yield improvements are terminated early, reducing unnecessary computational costs.
After extensive exploration of the hyperparameter space, `Optuna` identified the following optimal set of hyperparameters, which maximized the GNN's performance in terms of accuracy, at least for the flow cases under consideration, that will be discussed in Chapters 5, 6, and 7:

- Embedded dimension: 35. This hyperparameter controls the size of the hidden feature space for each node in the GNN. A higher dimension allows for richer representations of each node's features, but too high a value would increase computational cost without necessarily improving performance.
- Number of GNN layers: $k = 40$. This defines the depth of the GNN, which in turn determines how many message-passing steps are performed across the graph. A deeper network allows each node to aggregate information from further parts of the graph, capturing global structure. However, increasing the number of layers too much can lead to over-smoothing, where node features become indistinguishable, and computational costs rise unnecessarily.
- Update relaxation weight: $\alpha = 6 \times 10^{-1}$. This coefficient scales the contribution of each message-passing update, controlling how much new information influences the node's updated feature representation. A well-chosen value of α ensures that updates are neither too drastic nor too small, facilitating stable and effective learning.
- Loss function weight: $\gamma = 0.1$. This parameter controls the weighting of the loss terms from different layers in the GNN during training. By assigning higher weights to losses from later layers, we emphasize updates that incorporate information from a wider context, which can improve the model's ability to capture long-range dependencies in the graph.
- Learning rate: $LR = 3 \times 10^{-3}$. The learning rate governs the step size for gradient descent during the training process. A higher learning rate speeds up convergence but may result in unstable updates, while a lower learning rate ensures more stable learning but can slow down convergence.

These hyperparameters were optimized to achieve a balance between the model's accuracy and its computational efficiency. The final set of hyperparameters enables the GNN to effectively model the complex dynamics of fluid

flows, while maintaining computational feasibility as a surrogate model for CFD applications.

It is worth noting that these hyperparameters are inherently task-dependent and, in theory, should be re-optimized for each specific learning task to ensure optimal performance. However, the primary objective of this thesis is to demonstrate the methodology rather than to focus on the exact numerical outcomes for each case. Consequently, it was not necessary to perform a hyperparameter optimization for every individual scenario.

Re-optimization was conducted for certain learning tasks, and the results indicated that the hyperparameters remained largely consistent, with only minor variations across different tasks. This outcome suggests that the GNN architecture demonstrates inherent robustness to hyperparameter variations within the range of learning tasks addressed in this thesis. As a result, the proposed GNN structure exhibits significant resilience to changes in hyperparameters, enabling it to perform effectively and efficiently across a broad spectrum of CFD problems without requiring substantial hyperparameter adjustments.

4 - Adjoint Optimization

4.1 . Introduction to optimization methods

Optimization methods are mathematical techniques designed to find the best possible solution to a problem within a set of constraints. These methods are essential in fields such as engineering, economics, and data science, where improving efficiency, performance, or resource allocation is key important. The core of optimization lies in maximizing or minimizing an objective function, which represents the goal of the problem, subject to given constraints.
In the context of this thesis, optimization methods are introduced to provide the foundation for the adjoint methods, which are critical, in this context, for enforcing the RANS equations within the optimization loop of the GNN model.

4.2 . History of optimization

Figure 4.1: (Left) Johann Bernoulli (Basel, 1667 - Basel 1748) (Right) Leonhard Euler (Basel, 1707 – Saint Petersburg, 1783

The history of optimization reflects the broader evolution of mathematical thought and the increasing computational capabilities that have shaped its development over the centuries. In the 17th century, pioneers like Johann Bernoulli and Leonhard Euler (Fig. 4.1) began exploring optimization through the calculus of variations, applying it to physical problems. Their work laid the groundwork for formal optimization techniques.
In the 20th century, George Dantzig introduced the Simplex Method, a major breakthrough in solving linear programming problems. This method quickly found applications in transportation, manufacturing, and finance by optimizing linear objective functions under linear constraints. With the rise of computational power, non-linear optimization methods such as gradient descent and Newton's method were developed to handle more complex systems where both the objective functions and constraints are non-linear.
The latter half of the century saw the emergence of global optimization techniques like Genetic Algorithms [Goldberg and Holland, 1988], Simulated An-

nealing [Kirkpatrick et al., 1983], and Particle Swarm Optimization [Kennedy and Eberhart, 1995]. These metaheuristic approaches were designed to tackle problems with multiple local optima, inspired by natural processes and offering solutions for highly complex and multimodal problems.
In recent years, optimization has advanced further with machine learning and computational growth. Modern methods such as Bayesian optimization and Reinforcement Learning (RL) are now used to solve large-scale, high-dimensional problems, particularly in fields like artificial intelligence and data science.

4.3 . Fundamental concepts of optimization

Optimization problems involve determining the best possible solution by maximizing or minimizing an objective function, which mathematically represents the goal of the problem. Formally, the objective function is expressed as $f(\mathbf{x})$, where $\mathbf{x}$ represents the decision variables on which the function depends. For example, in a cost minimization problem, $f(\mathbf{x})$ could represent the total cost to be minimized, and $\mathbf{x}$ could be the set of variables influencing that cost.
In most optimization problems, solutions are subject to certain constraints, which are conditions that must be satisfied by the decision variables. Constraints can be written in the form of equalities (e.g., $g(\mathbf{x}) = 0$) or inequalities (e.g., $h(\mathbf{x}) \leq 0$) and represent the physical, financial, or operational limitations of the problem. These constraints define the feasible region, which is the set of all points $\mathbf{x}$ that satisfy the given constraints. The solution to the optimization problem must lie within this feasible region.
Depending on the structure of the objective function and the constraints, optimization problems can be classified into different categories:

- Linear Optimization: In linear optimization, both the objective function and the constraints are linear. A linear objective function can be expressed as:

$$f(\mathbf{x}) = \mathbf{C}\mathbf{x} \tag{4.1}$$

  where $\mathbf{C} \in \mathbb{R}^{m \times n}$ is the coefficients matrix and $\mathbf{x} \in \mathbb{R}^n$ is the decision variables vector. Similarly, the constraints are also linear functions of the decision variables. This class of problems can be efficiently solved using methods like the Simplex Method or Interior Point Methods.

- Nonlinear Optimization: In nonlinear optimization, either the objective function or the constraints (or both) are nonlinear, for example:

$$f(\mathbf{x}) = \mathcal{N}(\mathbf{x}) \tag{4.2}$$

where $\mathcal{N}(\mathbf{x})$ is a generic non-linear operator. Nonlinear problems are inherently more complex and often require iterative methods like Gradient Descent, Newton's Method, or Quasi-Newton Methods. These algorithms are commonly used in machine learning, engineering, and scientific research.

A special class of optimization problems arises when the objective function is convex and the feasible region is a convex set. Convexity is a critical property that simplifies the optimization process. In convex optimization, if a function $f(\mathbf{x})$ is convex and the feasible region is also convex, any local minimum is guaranteed to be a global minimum. Mathematically, a function $f(\mathbf{x})$ is convex if:

$$f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2) \tag{4.3}$$

for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$ and $\lambda \in [0,1]$. This property greatly reduces the complexity of finding an optimal solution, as it avoids the problem of multiple local minima often present in non-convex optimization.

For a solution to be considered optimal, certain conditions must be satisfied. These conditions vary depending on whether the problem is unconstrained or constrained.

- Unconstrained Optimization: In the simplest case, the goal is to find the minimum or maximum of an objective function $f(\mathbf{x})$ without any constraints. If the function is differentiable, a necessary condition for $\mathbf{x}$ to be a local minimum, called $\mathbf{x}^{opt}$, is that the gradient of f at $\mathbf{x}^{opt}$ must be zero:

$$\nabla f(\mathbf{x}^{opt}) = 0 \tag{4.4}$$

  This condition indicates that the function has a stationary point at $\mathbf{x}^{opt}$. However, this is only a necessary condition. To further ensure that $\mathbf{x}^{opt}$ is a local minimum, a second-order condition must also be satisfied: the Hessian matrix of $f(\mathbf{x})$, denoted by $\mathbf{H}_f(\mathbf{x}^{opt})$, must be positive semi-definite at $\mathbf{x}^{opt}$. This ensures that the function curves upwards in all directions at $\mathbf{x}^{opt}$, confirming it is a local minimum.

- Constrained Optimization: When constraints are present, the optimality conditions become more complex. In this case, the Karush-Kuhn-Tucker (KKT) conditions provide the necessary criteria for optimality. The KKT conditions extend the concept of the gradient to account for the constraints by introducing Lagrange multipliers.

4.4 . Optimization methods overview

Optimization problems can be broadly categorized into gradient-based methods, gradient-free methods, and hybrid methods. Each category ad-

dresses different types of optimization problems by employing various mathematical and computational strategies to search for optimal solutions. The choice of method depends on the properties of the objective function, such as smoothness, differentiability, and the presence of constraints, as well as the computational resources available.

4.4.1 . Gradient based methods

Gradient-based methods rely on the gradient of the objective function to iteratively move towards an optimal solution. These methods assume that the objective function is differentiable, and they make use of first-order (and sometimes second-order) derivative information to guide the search. Gradient-based techniques are effective when the function is smooth and well-behaved, as they can exploit local curvature information to accelerate convergence.

- Gradient Descent: The Gradient Descent method is one of the simplest and most widely used optimization algorithms. It iteratively adjusts the solution based on the gradient of the objective function, moving in the direction of steepest descent to minimize the function. Starting from an initial guess x_0, the update rule is given by:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k) \tag{4.5}$$

  where α is the step size or learning rate and $\nabla f(\mathbf{x}_k)$ is the gradient of the objective function evaluated at $\mathbf{x}_k$. The algorithm continues until the gradient is sufficiently small, indicating convergence to a local minimum. An important variant is Stochastic Gradient Descent (SGD), which approximates the gradient using a random subset (or mini-batch) of data points, making it particularly effective for large-scale problems like those encountered in machine learning. The use of smaller batches reduces computational overhead per iteration, but it may introduce noise into the gradient, which can slow convergence but also help escape local minima.

- Newton's Method: Newton's method incorporates second-order derivative information through the Hessian matrix of the objective function. It is particularly effective when the objective function is smooth and convex, as it adjusts the step direction and size based on the curvature of the function. The update rule for Newton's method is:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}_f(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k) \tag{4.6}$$

  where $\mathbf{H}_f(\mathbf{x}_k)$ is the Hessian matrix, which represents the second-order partial derivatives of the objective function at $\mathbf{x}_k$. This method converges faster than gradient descent, especially near the optimal point, but requires calculating and inverting the Hessian, which can be computationally expensive for large problems.

- Quasi Newton Method: Quasi-Newton methods aim to approximate the behavior of Newton's method without directly computing the Hessian matrix. Instead, they build an approximation to the Hessian using only gradient information. One of the most widely used quasi-Newton methods is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm. The BFGS method iteratively updates an estimate of the Hessian inverse, making it more efficient for high-dimensional problems compared to standard Newton's method. A limited-memory version, L-BFGS, is often used for very large-scale problems, as it reduces memory and computational requirements by storing only a few vectors from the gradient history.

- Conjugate Gradient Method: The conjugate gradient method is particularly suited for large-scale optimization problems, especially those involving large, sparse systems. Conjugate gradient methods generate a sequence of k search directions that are mutually conjugate with respect to the Hessian $\mathbf{H}_f$, meaning that the following condition has to be satisfied by the k-th search direction $\mathbf{p}_k$:

  $$\mathbf{p}_k^T \mathbf{H}_f \mathbf{p}_{k-1} = 0. \tag{4.7}$$

  This method is commonly used for solving large linear systems but can also be applied to non-linear optimization when coupled with line search techniques. The next step optimization process is obtained as:

  $$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \tag{4.8}$$

  where α_k is the step size along the k-th search direction.

4.4.2 . Gradient free methods

In situations where the objective function is non-differentiable, discontinuous, or noisy, or computation of gradients is computationally expensive, gradient-based methods become impractical or ineffective. Gradient-free methods are designed for such cases, as they do not require derivative information and instead rely on function evaluations.
The key techniques in this category include (non-exhaustive list):

- Genetic Algorithms (GA): are inspired by the principles of natural selection and genetics. They work by evolving a population of candidate solutions over several iterations (generations). At each generation, the best solutions are selected, combined (crossover), and randomly mutated to explore new areas of the solution space. Over time, the population converges toward optimal or near-optimal solutions. Genetic Algorithms are highly flexible and can be applied to a wide range of optimization problems, especially those with complex, multimodal landscapes.

- Simulated Annealing (SA): is inspired by the annealing process in metallurgy, where a material is slowly cooled to remove defects and minimize energy. In optimization, SA probabilistically accepts worse solutions early in the search to avoid local minima, but gradually reduces the acceptance probability as the search progresses. This approach allows the algorithm to explore the solution space more broadly at the start and then focus on fine-tuning the best solutions as it converges.
- Bayesian Optimization: is a sequential design strategy for the global optimization of black-box functions, where the objective function is expensive to evaluate. It builds a probabilistic model (usually a Gaussian process) of the objective function and uses this model to decide where to sample next. By balancing exploration (sampling regions with high uncertainty) and exploitation (sampling regions likely to have good solutions), Bayesian optimization can efficiently find optimal solutions with a limited number of function evaluations.

4.5 . Introduction to Adjoint Methods

In the context of optimization, especially for high-dimensional and computationally expensive problems, *Adjoint Methods* provide an efficient way to compute gradients of an objective function with respect to a large number of variables. These methods are particularly valuable when solving optimization problems that involve PDEs, such as the RANS equations in CFD.
Adjoint methods have been foundational in CFD optimization, notably through the pioneering contributions by Jameson [1988], who pioneered adjoint-based optimization for aerodynamic design, and by Pironneau [1974], who explored optimal design approaches in fluid mechanics, laying the groundwork for adjoint applications in CFD. Amoignon et al. [2004] extended these ideas by applying adjoint-based techniques for the shape optimization of aerodynamic surfaces, specifically targeting natural laminar flow designs. Giles and Pierce [2000], furthered this development, providing a comprehensive introduction to the adjoint approach specifically tailored to design problems in CFD. Optimization problems with PDE constraints have also received considerable attention. Works by Borzi and Schulz [2012] and Hinze et al. [2008] address the complexities involved in optimization constrained by elliptic and parabolic PDEs, while Lions [1971] introduced foundational concepts for handling hyperbolic constraints in optimization frameworks.
Within this context, the Field Inversion and Machine Learning (FIML) framework proposed by Parish and Duraisamy [2016] is particularly relevant. This approach combines adjoint-based optimization with machine learning to infer unknown quantities, such as closure terms in turbulence modeling, directly from data. The methodology shares conceptual similarities with the scope of

this thesis, where adjoint methods are used to guide the training of ML models while ensuring consistency with governing PDEs.
Based on these fundamental works, adjoint equations have been also extensively used in the context of flow control, flow sensitivity and instability [Cherubini et al., 2010, 2013, Semeraro et al., 2013, Loiseau et al., 2014]. Among the others, it is worth mentioning the works by Giannetti and Luchini [2007], Marquet et al. [2008] and Luchini et al. [2009] on the sensitivity framework, providing a method to identify critical regions in the flow most susceptible to control interventions. This technique has been later extended to numerous works and configurations, ranging from bluff bodies wakes to control of Rijke tube oscillations [Luchini and Bottaro, 2014]. These approaches allow targeted adjustments that can suppress instability or delay transition by carefully manipulating the sensitive regions identified by adjoint solutions. Recent developments discussed in Costanzo et al. [2022] may enable the application of these techniques to cases characterized by larger computational domains or unsteady flows through parallel-in-time optimization.
An extensive review covering many of these applications is provided by Luchini and Bottaro [2014], who highlight how adjoint-based techniques have transformed our understanding of flow stability and receptivity, particularly in cases like the noise-amplifying instabilities in boundary layers.

The applications of adjoint methods extend beyond stability analysis. For instance, they have become crucial in error estimation and grid adaptation within CFD, as noted by Venditti and Darmofal [2003] and Park [2002], who demonstrate how adjoint error analysis can guide grid refinement to improve accuracy in functional outputs like drag or lift. The importance of this approach is underscored by Rumsey and Ying [2002] for high-lift configurations, where numerical errors and modeling fidelity significantly affect simulation results. Furthermore, uncertainty quantification has leveraged adjoint-based sensitivity to manage the *curse of dimensionality*, allowing efficient propagation of uncertainties through complex CFD simulations. For example, Wang et al. [2009] and Dow and Wang [2013] have used adjoint sensitivity analysis to identify the most influential variables affecting turbomachinery performance, while facilitating surrogate model development to address geometric variability.

In this thesis, Adjoint Methods are crucial because they enable us to enforce the physical constraints defined by the RANS equations within the training process of the GNN. By integrating adjoint-based optimization, we can effectively compute the gradients required to adjust the parameters of the GNN. This makes the optimization process physically consistent, ensuring that the predictions of the GNN align with fluid behavior.

4.5.1 . General Methodology of Adjoint Methods

The central idea behind Adjoint Methods is to reformulate optimization problems constrained by partial differential equations (PDEs). These methods leverage the fact that the governing constraints, expressed as PDEs such as the RANS equations, are inherently satisfied by the solver, without requiring them to be explicitly enforced in the optimization process. This allows the introduction of adjoint variables (or Lagrange multipliers), which are chosen to simplify the gradient computation. With reference to a general optimization problem, we introduce an objective function $J(\mathbf{x})$, which we aim to minimize (or maximize), subject to constraints represented by a set of PDEs

$$\min_{\mathbf{x}} \; J(\mathbf{x}, \mathbf{u}(\mathbf{x})) \quad \text{subject to} \quad C(\mathbf{u}(\mathbf{x}), \mathbf{x}) = 0 \tag{4.9}$$

where $J(\mathbf{x}, \mathbf{u}(\mathbf{x}))$ is the objective function that depends on both the design variables $\mathbf{x}$ and the state variables $\mathbf{u}(\mathbf{x})$, which typically satisfy the governing PDEs. $C(\mathbf{u}(\mathbf{x}), \mathbf{x}) = 0$ represents the PDE constraint, where C is a set of governing equations (e.g., RANS equations) that describe the behavior of the state variables $\mathbf{u}$ given the design variables $\mathbf{x}$.
The key challenge here is that directly computing the gradient of J with respect to $\mathbf{x}$ would require solving the PDE for every design variable, which is computationally prohibitive for high-dimensional problems. To overcome this issue, the adjoint method introduces the Lagrangian functional:

$$\mathcal{L}(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}) = J(\mathbf{x}, \mathbf{u}) + \boldsymbol{\lambda}^T C(\mathbf{u}, \mathbf{x}), \tag{4.10}$$

where $\boldsymbol{\lambda}$ is the vector of adjoint variables (or Lagrange multipliers). The PDE constraints $C(\mathbf{u}, \mathbf{x}) = 0$ are inherently satisfied by the solver ensuring that the term $\boldsymbol{\lambda}^T C(\mathbf{u}, \mathbf{x})$ is identically zero. This provides the flexibility to choose $\boldsymbol{\lambda}$ in a way that simplifies the gradient computation.
By constructing the Lagrangian functional, we convert the constrained optimization problem into an unconstrained one, where the solution is sought by zeroing the variations of $\mathcal{L}$ with respect to both the state variables $\mathbf{u}$ and the adjoint variables $\boldsymbol{\lambda}$. To compute the gradient of J with respect to the design variables $\mathbf{x}$, we proceed by taking the total derivative of $\mathcal{L}$ with respect to $\mathbf{x}$:

$$\frac{d\mathcal{L}}{d\mathbf{x}} = \frac{\partial J}{\partial \mathbf{x}} + \frac{\partial J}{\partial \mathbf{u}} \frac{d\mathbf{u}}{d\mathbf{x}} + \boldsymbol{\lambda}^T \left(\frac{\partial C}{\partial \mathbf{x}} + \frac{\partial C}{\partial \mathbf{u}} \frac{d\mathbf{u}}{d\mathbf{x}} \right) \tag{4.11}$$

In this expression, $\frac{d\mathbf{u}}{d\mathbf{x}}$ represents how the state variables change with respect to the design variables. However, solving for $\frac{d\mathbf{u}}{d\mathbf{x}}$ directly is expensive, as it involves differentiating the entire set of PDEs with respect to $\mathbf{x}$. To avoid this, we first take the variation of the Lagrangian functional with respect to $\mathbf{u}$, enforcing the following *adjoint equation* to be satisfied:

$$\frac{d\mathcal{L}}{d\mathbf{u}} = \frac{\partial J}{\partial \mathbf{u}} + \boldsymbol{\lambda}^T \frac{\partial C}{\partial \mathbf{u}} = 0. \tag{4.12}$$

Solving this equation provides the adjoint variable $\boldsymbol{\lambda}$. With $\boldsymbol{\lambda}$ determined, the gradient of the objective function with respect to the design variables $\mathbf{x}$ can be simplified to:

$$\frac{d\mathcal{L}}{d\mathbf{x}} = \frac{\partial J}{\partial \mathbf{x}} + \boldsymbol{\lambda}^T \frac{\partial C}{\partial \mathbf{x}}. \tag{4.13}$$

This is the central result of the adjoint method: the gradient $\frac{d\mathcal{L}}{d\mathbf{x}}$ is computed without needing to evaluate $\frac{d\mathbf{u}}{d\mathbf{x}}$ directly, which significantly reduces the computational cost.

To summarize the entire adjoint process:

1. Solve the PDE (primal problem): For a given design variable $\mathbf{x}$, solve the governing equations $C(\mathbf{u}(\mathbf{x}), \mathbf{x}) = 0$ to find the state variables $\mathbf{u}(\mathbf{x})$.

2. Solve the adjoint equation: Using the solution for $\mathbf{u}(\mathbf{x})$, solve the adjoint equation to find the adjoint variable $\boldsymbol{\lambda}$.

3. Compute the gradient: With $\boldsymbol{\lambda}$ and $\mathbf{u}(\mathbf{x})$, compute the gradient of the objective function with respect to the design variables $\mathbf{x}$.

4. Update the design variables: Use an optimization algorithm (e.g., gradient descent, Newton's method) to update $\mathbf{x}$ based on the computed gradient.

By repeating these steps, the adjoint method efficiently finds the optimal solution to high-dimensional optimization problems constrained by complex PDEs.

In this thesis, adjoint methods are employed to compute the gradients necessary for training the GNN. Instead of relying solely on the differentiable nature of the GNN to calculate the gradients from numerical data, the adjoint method is used to obtain analytic gradients that are grounded in the RANS, which govern fluid dynamics. This approach ensures that the gradients incorporate the underlying physical laws, rather than being based purely on data. By leveraging adjoint methods, the GNN is trained to not only minimize prediction error but also to adhere to the RANS equations, resulting in a model that is both data-driven and physically consistent.

4.5.2 . Adjoint Method applied to RANS

In this section, the Adjoint Method is tailored for the CFD applications analyzed in this thesis. While the general principles of Adjoint Methods have already been discussed, this section introduces the specific details of the custom Adjoint Method we developed for our case study. The application draws inspiration from the work of Foures et al. [2014], where a data assimilation scheme is introduced based on this framework; specifically, the baseline model

is provided by the RANS equations (Eq. 2.8b) and the control variable is represented by the forcing stress term $\mathbf{f}$ (Eq. 2.9). The goal is to minimize the discrepancy between a reconstructed mean flow field $\hat{\overline{\mathbf{u}}}$ and a ground truth mean flow $\overline{\mathbf{u}}$ (Eq. 2.7) obtained from numerical simulations.
At the core of any adjoint-based optimization method lies the definition of a cost function that quantifies the difference between the desired and predicted results. In our case, the goal of the optimization is to minimize the error between the ground truth mean flow $\overline{\mathbf{u}}$ (obtained from high-fidelity simulations) and the reconstructed mean flow $\hat{\overline{\mathbf{u}}}$ produced by the RANS model during the optimization loop. The cost function that measures this discrepancy is defined as:

$$\varepsilon\left(\hat{\overline{\mathbf{u}}}\right) = \frac{1}{2}||\overline{\mathbf{u}} - \hat{\overline{\mathbf{u}}}||_2 \tag{4.14}$$

where $||\cdot||_2$ represents the L^2-norm, defined as

$$||\mathbf{c}||_2 = \sqrt{\langle \mathbf{c} \cdot \mathbf{c} \rangle}, \tag{4.15}$$

and associated with the scalar product

$$\langle \mathbf{a}, \mathbf{b} \rangle = \int_\Omega \mathbf{a} \cdot \mathbf{b} d\Omega, \tag{4.16}$$

with $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ denoting arbitrary vectors and Ω denoting the computational domain. The smaller the value of ε, the closer the predicted flow $\hat{\overline{\mathbf{u}}}$ is to the ground truth $\overline{\mathbf{u}}$.
The control variable in our optimization framework is the forcing stress term $\mathbf{f}$. This forcing term is introduced into the RANS equations (Eq. 2.8b) as a means to adjust the flow predictions $\hat{\overline{\mathbf{u}}}$ in order to minimize the cost function. The ultimate goal of the optimization process is to iteratively refine this forcing term so that the predicted flow $\hat{\overline{\mathbf{u}}}$ matches the ground truth flow as closely as possible. However, since the cost function ε does not directly depend on the forcing stress $\mathbf{f}$ (Eq. 2.9), a relationship between the two must be established in order to compute the necessary gradients for optimization. To this end, an augmented Lagrangian functional is introduced, that incorporates both the objective function and the governing constraints (i.e., the RANS equations). The augmented Lagrangian formalism allows to transform a constrained optimization problem into an unconstrained one. This is achieved by incorporating the governing equations of the system (in this case, the RANS equations) into the optimization framework via Lagrange multipliers. For our case, the augmented Lagrangian functional is defined as:

$$\mathcal{L}\left(\mathbf{f}, \hat{\overline{\mathbf{u}}}, \hat{p}, \hat{\overline{\mathbf{u}}}^\dagger, \hat{p}^\dagger\right) = \varepsilon\left(\hat{\overline{\mathbf{u}}}\right) - \langle \hat{\overline{\mathbf{u}}}^\dagger, \hat{\overline{\mathbf{u}}} \cdot \nabla \hat{\overline{\mathbf{u}}} + \nabla \hat{p} - \frac{1}{Re} \nabla^2 \hat{\overline{\mathbf{u}}} - \mathbf{f} \rangle - \langle \hat{p}^\dagger, \nabla \cdot \hat{\overline{\mathbf{u}}} \rangle. \tag{4.17}$$

In this expression, $\hat{\overline{\mathbf{u}}}^\dagger$ and $\hat{p}^\dagger$ are the adjoint variables (also called Lagrange multipliers), introduced to compute the gradients efficiently. These adjoint

variables are fundamental to the adjoint method, as they ensure that the cost function minimizes with respect to both the direct variables ($\hat{\overline{\mathbf{u}}}$ and $\hat{p}$) and the control variable ($\mathbf{f}$). The terms $\langle \cdot, \cdot \rangle$ represent spatial scalar products as defined in Eq. 4.16, integrating the Lagrange multiplier terms over the domain of the flow.

By minimizing the augmented Lagrangian functional, we derive the adjoint NSE. These equations govern the behavior of the adjoint variables and are essential for computing the gradient of the cost function with respect to the control variable $\mathbf{f}$. The adjoint NSE are given by:

$$-\hat{\overline{\mathbf{u}}} \cdot \nabla \hat{\overline{\mathbf{u}}}^{\dagger} + \hat{\overline{\mathbf{u}}}^{\dagger} \cdot \nabla \hat{\overline{\mathbf{u}}}^{T} - \nabla \hat{p}^{\dagger} - \frac{1}{Re} \nabla^2 \hat{\overline{\mathbf{u}}}^{\dagger} = \frac{\partial \varepsilon}{\partial \hat{\overline{\mathbf{u}}}} \tag{4.18a}$$

$$\nabla \cdot \hat{\overline{\mathbf{u}}}^{\dagger} = 0. \tag{4.18b}$$

The first equation governs the momentum equation for the adjoint variables, while the second enforces the incompressibility condition on the adjoint velocity field. These adjoint equations are forced by the derivative of the cost function ε with respect to the predicted mean flow $\hat{\overline{\mathbf{u}}}$. This term, $\frac{\partial \varepsilon}{\partial \hat{\overline{\mathbf{u}}}}$, can be computed directly from the cost function as:

$$\frac{\partial \varepsilon}{\partial \hat{\overline{\mathbf{u}}}} = \hat{\overline{\mathbf{u}}} - \overline{\mathbf{u}}. \tag{4.19}$$

This equation expresses the difference between the predicted mean flow and the ground truth mean flow, which drives the adjoint optimization process. It provides the necessary gradient information for adjusting the forcing term $\mathbf{f}$ to minimize the cost function ε.

Once the adjoint equations have been solved, we can compute the gradient of the cost function with respect to the control variable $\mathbf{f}$. This gradient is given by:

$$\frac{\partial \varepsilon}{\partial \mathbf{f}} = \hat{\overline{\mathbf{u}}}^{\dagger}. \tag{4.20}$$

This equation provides the information needed to update the forcing term $\mathbf{f}$ during the optimization process. The adjoint variable $\hat{\overline{\mathbf{u}}}^{\dagger}$ acts as the sensitivity of the cost function to changes in the forcing term. By using this gradient, we can iteratively adjust the forcing term to minimize the cost function and bring the predicted flow closer to the ground truth. The optimization process proceeds by updating the forcing term $\mathbf{f}$ according to a gradient descent algorithm:

$$\mathbf{f}^{(n+1)} = \mathbf{f}^{(n)} - \frac{\partial \varepsilon^{(n)}}{\partial \mathbf{f}^{(n)}} \tag{4.21}$$

where n denotes the iteration number. The optimization loop continues until the cost function ε reaches an acceptable threshold, indicating that the predicted flow $\hat{\overline{\mathbf{u}}}$ closely matches the ground truth $\overline{\mathbf{u}}$.

The boundary conditions for the adjoint NSE differ from those of the direct problem, as they reflect the sensitivity of the cost function to perturbations at the boundaries. Nonetheless, since the adjoint boundary conditions rely on the direct problem's boundary conditions, these latter are included here for reference and clarification

$$\begin{cases} u = 1, \;\; v = 0 & \text{at the inlet,} \\ u = 0, \;\; v = 0 & \text{on the cylinder surface,} \\ \partial_y u = 0, \;\; v = 0 & \text{on symmetry boundaries,} \\ \dfrac{1}{Re}\partial_x u - p = 0, \;\; \partial_x v = 0 & \text{at the outlet.} \end{cases} \tag{4.22}$$

By performing integration by parts and eliminating terms that depend on boundary conditions, we derive the boundary conditions for the adjoint problem directly from those of the direct problem. For our adjoint method, this leads to the following boundary conditions for the adjoint velocity $\hat{\overline{\mathbf{u}}}^\dagger$ and adjoint pressure $\hat{p}^\dagger$:

$$\begin{cases} u^\dagger = 1, \;\; v^\dagger = 0 & \text{at the inlet,} \\ u^\dagger = 0, \;\; v^\dagger = 0 & \text{on the cylinder surface,} \\ \partial_y u^\dagger = 0, \;\; v^\dagger = 0 & \text{on symmetry boundaries,} \\ \dfrac{1}{Re}\partial_x u^\dagger + p^\dagger = -uu^\dagger, \;\; \dfrac{1}{Re}\partial_x v^\dagger = -uv^\dagger & \text{at the outlet.} \end{cases} \tag{4.23}$$

These boundary conditions ensure that the adjoint variables are appropriately constrained at the domain boundaries, allowing for accurate computation of the gradient $\dfrac{\partial \varepsilon}{\partial \mathbf{f}}$.
To summarize the complete optimization loop, the key steps are listed in the following.

- Initialization: An initial guess for the control variable $\mathbf{f}$ is made. Typically, $\mathbf{f} = 0$ is chosen as the starting point to satisfy divergence-free and no-slip conditions.
- Forward Step: The direct RANS equations are solved to obtain a prediction of the mean flow $\hat{\overline{\mathbf{u}}}$, given the current forcing term $\mathbf{f}$.
- Cost Function Evaluation: The cost function ε is evaluated, measuring the error between the predicted flow $\hat{\overline{\mathbf{u}}}$ and the ground truth $\overline{\mathbf{u}}$.
- Adjoint Step: The adjoint NSE are solved to compute the adjoint variables $\hat{\overline{\mathbf{u}}}^\dagger$ and $\hat{p}^\dagger$.
- Control Variable Update: Using the adjoint variables, the gradient of the cost function with respect to the forcing term is computed, and $\mathbf{f}$ is updated accordingly.

The loop is repeated until convergence, *i.e.* when the cost function ε falls below a chosen threshold.

The adjoint method developed for this thesis handles the optimization of the forcing term $\mathbf{f}$ within the RANS framework. By combining the adjoint-based gradients, this data assimilation scheme can be coupled to the training process of the GNN in order to ensure that the model is trained in a manner consistent to the simulation data and the governing laws. This physically consistent approach enhances the predictive accuracy of the model while minimizing computational costs, making it a robust tool for CFD applications.

5 - Part I: RANS closure term prediction

5.1 . Introduction

The main goal of the present work is to develop a GNN-based surrogate model to predict the closure term $\mathbf{f}$ of a RANS system of equations, given as input the meanflow.
In a very schematic way, for a regression problem, ML techniques enable to identify mappings between observables of a system (inputs) and quantities of interest (outputs) we aim to predict by leveraging data; when these analyzed data are governed by deterministic or statistical laws, in principle, these mappings correspond to approximating models.
At low Reynolds numbers, we consider a data assimilation approach, where the closure model corresponds to the control parameter of an adjoint-based loop [Foures et al., 2014]; without explicitly introducing a tensor structure or Boussinesq approximations, this method is well suited for non-homogeneous flows at lower Reynolds numbers. Here, we take inspiration from these applications and mainly focus on unsteady flows developing past bluff bodies at low Reynolds numbers $50 \leq Re \leq 150$; we consider RANS as baseline, although alternative choices can be also adopted, such as Euler equations or linearized NSE in resolvent form, where the parameter to be tuned is the dissipation term [Morra et al., 2019, Pickering et al., 2021, von Saldern et al., 2023].
With respect to standard approaches, we focus here on a supervised learning strategy where the closure model is identified by inference from available data. In principle, we could identify universal closure models directly from data, having at disposal an infinite amount of them. In practice, in real cases, we may deal with a limited amount of data or few localized measurements; these limitations can impact on the use of methods such as NNs, where the expressivity and generic structure make them suitable for a large class of models, but prone to generalization problems. In this sense, ML models risk to be representative solely of the datasets included in the training process; thus, it becomes compelling given the available data to circumvent these problems by inputting well selected data during the training or providing prior knowledge through modelling [Shukla et al., 2022, Bucci et al., 2023].

With this in mind, we will test if it is possible to identify generic closure models from data defined on unstructured meshes while only relying on a small amount of data chosen on principled criteria. In order to answer these questions, a first ingredient is the introduction of GNN [Scarselli et al., 2008]; this architecture is characterized by complex multi-connected nets of nodes that can be naturally adapted to unstructured meshes: the convolution in a GNN is performed by aggregating information from neighboring nodes, thus

overcoming the limitations imposed by the geometry in contrast with CNN. Moreover, GNNs: i) show remarkable generalization capabilities as compared to standard network models [Sanchez-Gonzalez et al., 2018]; ii) are differentiable; iii) provide the possibility of directly targeting the learning of the operator via discrete stencils [Shukla et al., 2022]. Due to these features, this architecture has recently attracted attention in fluid mechanics. A review is available on the subject authored by Lino et al. [2023], while examples are given by the works of Toshev et al. [2023] or Dupuy et al. [2023b], where wall shear stress are modelled for LES simulations based on GNN. Here, we take inspiration from Donon et al. [2020], where a GNN-based architecture incorporating permutation and translation invariance is combined with the statistical solver problem; Donon et al. [2020] proofed that the architecture – referred to as deep statistical solver – has some universal approximation properties, and it is capable of operator learning.

In this contribution, GNN are combined with numerical simulations performed using FEM. The GNN-FEM interface allows the use of NN predictions in post-processing FEM analysis since it provides a two-way coupling between NN and FEM environments. However – as already mentioned – the volume of data can represent a bottleneck if the available amount is insufficient. Moreover, also quality of data impacts on the prediction properties in pure data-driven modelling [Bucci et al., 2023], in particular when data at hand are not sufficient in representing correctly the distribution of the overall dataset. Inconsistent or unbalanced data distributions may lead the model to being trained on subsets of data that do not adequately represent the underneath physics of the problem. Thus, as second, fundamental ingredient we adopt *active learning*, aimed to increase the dissimilarity between data points and ensure that the model distills all discriminant features necessary to perform the required task effectively. At the best of our knowledge, this application is among the first in fluid mechanics where selection of data is performed by applying an active learning criterion. Readers are directed to Ren et al. [2021] for a comprehensive overview of active learning methods in deep learning. We follow the work by Charpiat et al. [2019] where scalar products of gradients associated with the update of the model weights of the GNN are considered as similarity metric between samples; the chosen criterion allows to dynamically increase the training dataset by introducing data that promote diversity in the dataset.

The set of equations used for the numerical simulations is discussed in Sec. 2.3.3, while details on the numerical setup can be found in Sec. 5.2.1. The flow cases are shown in Sec. 5.2. The theoretical background and how this architecture is coupled with numerical simulations resolved on unstructured meshes can be found in Sec. 3.8. Then, the work moves into the results section (Sec. 5.3): GNN models are trained on datasets composed by the

mean flows (input) and the Reynolds stress (target); the baseline is provided by the closure problem in cylinder flows at Reynolds numbers $50 \leq Re \leq 150$ (Sec. 5.3.2). Generalization properties are assessed using 4 different benchmarks defined in Sec. 5.3.1, not included in the training sets. This baseline is thus compared with models trained with different strategies of data selection. First, we consider cases where data augmentation is performed using simulations of flows past bluff bodies of random geometry (Sec. 5.3.3); second, active learning is introduced in Sec. 5.3.3. Some conclusions on the work presented in this Chapter are drawn in Sec. 5.4.

5.2 . Design of experiment and numerical setup

The numerical simulations used to generate the ground truth data for training the GNN were conducted in-house using a custom `Python` script, built on the `FEniCS` library [Alnæs et al., 2015]. This computational framework was specifically designed to solve the PDEs equations such as fluid dynamics related problems using the FEM approach. In this study, we consider bluff bodies geometries at different Reynolds numbers, such that a comprehensive dataset is obtained to handle a larger variety of flow scenarios. In the following, the reference case – *i.e.* the cylinder flow – is discussed in Sec. 5.2.1 along with the necessary numerical details, while the flows past random geometries are discussed in Sec. 5.2.2.

5.2.1 . Cylinder flow

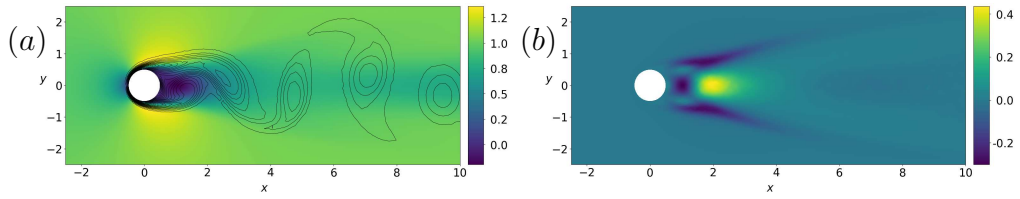


Figure 5.1: (a) Stream-wise component of the meanflow $\overline{\mathbf{u}}$ and vorticity isolines $\omega = \nabla \times \mathbf{u}$ for the flow past a cylinder at $Re = 150$. (b) For the same case, the stream-wise component of the closure term $\mathbf{f}$ is shown. In both cases, only a portion of the domain is shown.

The unsteady wake developing past a cylinder is a well documented benchmark in fluid dynamics literature, and it is often found as reference case in works of data-assimilation for the assessment of novel techniques; it exhibits steady behavior until a critical Reynolds number of $Re_c \cong 46.7$, when a supercritical Hopf bifurcation occurs [Provansal et al., 1987, Giannetti and Luchini, 2007]. At higher Reynolds numbers, the baseflow becomes an unsta-

ble solution and the unsteady flow develops into a limit cycle known as von Karman street (Fig. 5.1a). This behavior can be observed until $Re = 150$, for two-dimensional (2D) cases. Beyond this point, alongside periodic vortex formation, irregular velocity fluctuations begin to emerge [Anatol, 1958]; note that transition to turbulence occurs when three dimensional cases are considered, which is beyond the scope of the present work. Instead, this work aims at a quantitative analysis of GNN model emulators trained in low-data limits leveraging active learning processes; as such, we will benchmark the proposed algorithm on simpler 2D scenarios exhibiting the limit cycle behavior, in the range $50 \leq Re \leq 150$; we perform numerical simulations at different Re numbers in the mentioned range, with $\Delta Re = 10$, in order to collect the necessary data to train the GNN model and compose the dataset. The numerical simulations are resolved in time, while the meanflow $\overline{\mathbf{u}}$ and the forcing term $\mathbf{f}$ (Eq. 2.9) are computed by averaging on-the-fly until convergence. As an example of this data, Fig. 5.1a illustrates the stream-wise component of the meanflow $\overline{\mathbf{u}}$ alongside the vorticity isolines $\omega = \nabla \times \mathbf{u}$, while Fig. 5.1$b$ shows the stream-wise component of the closure term $\mathbf{f}$ for $Re = 150$. The key numerical aspects of the simulations, including the employed spatial discretization, time integration schemes, and solver techniques, are detailed in Guégan [2022]. Here, we provide a summary of the essential elements and highlight the validation process that underlines the reliability of the simulation data.

From a numerical perspective, the spatial discretization follows the finite element framework, using a weak formulation of the NSE. Specifically, the Taylor-Hood elements are employed, which use second-order (P2) elements for the velocity field and first-order (P1) elements for the pressure field. This choice ensures both the stability and accuracy of the solution, particularly for incompressible flow simulations, which are the primary focus of this work.
The time-stepping is handled via a second-order Backward Differentiation Formula (BDF), which ensures temporal accuracy:

$$\begin{cases} \left(\dfrac{3\mathbf{u}^n - 4\mathbf{u}^{n-1} + \mathbf{u}^{n-2}}{2\Delta t}\right) + (\mathbf{u}^{n-1}\cdot\nabla)\mathbf{u}^n + (\mathbf{u}^n\cdot\nabla)\mathbf{u}^{n-1} \\ \qquad -(\mathbf{u}^{n-1}\cdot\nabla)\mathbf{u}^{n-1} - \dfrac{1}{Re}\Delta\mathbf{u}^n + \nabla p^n = \mathbf{0} \\ \qquad\qquad\qquad\qquad\qquad\qquad \nabla\cdot\mathbf{u}^n = 0, \end{cases} \tag{5.1}$$

In this formulation the n superscript refers to the current time step, with $n-1$ and $n-2$ representing values from the previous two time steps. The time step Δt satisfies the CFL condition, ensuring numerical stability with $CFL \leq 0.5$ for all fluid flow cases under consideration.
The convective terms are handled using a combination of Newton and Picard iterative methods, where each iteration solves a linear system to converge to the final solution at each time step.

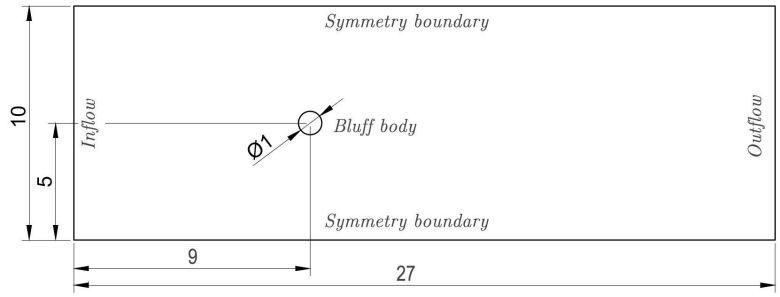


Figure 5.2: Sketch of the computational domain geometry. The diameter of the circumscribed circle of the bluff body, the height and length of the domain are given in non-dimensional units.

The mesh used in the simulations is unstructured and refined around critical flow regions, such as the area near the obstacle and the wake behind it. This mesh refinement ensures that the simulation accurately captures important flow features, including instabilities and vortex shedding. The reference mesh is based on the well-known benchmark case of flow around a cylinder, which serves as the foundation for validating the solver and ensuring the accuracy of the results.
Concerning the reference numerical setup, the characteristic dimension is the diameter D of the circumscribed circle to the bluff body. Based on this dimension, the computational domain extends $L_x = 27$ units in the streamwise direction and $L_y = 10$ units in the transverse direction. The system's origin $O(0,0)$ is positioned $\Delta x = 9$ units downstream from the inlet and $\Delta y = 5$ units from the symmetry boundaries. A pictorial sketch of the geometric configuration of the computational domain is reported in Fig. 5.2. The flow evolves from left to right with a dimensionless uniform velocity $\mathbf{u} = (1,0)^T$, normalized by the reference velocity U_∞ of the undisturbed flow. Boundary conditions follow the setup described by Foures et al. [2014], which read as:

$$\begin{cases} u = 1, \;\; v = 0 & \text{at the inlet,} \\ u = 0, \;\; v = 0 & \text{on the cylinder surface,} \\ \partial_y u = 0, \;\; v = 0 & \text{on symmetry boundaries,} \\ \dfrac{1}{Re}\partial_x u - p = 0, \;\; \partial_x v = 0 & \text{at the outlet.} \end{cases} \tag{5.2}$$

Validation of the numerical solver was conducted by comparing the simulated drag coefficient (C_d) and lift coefficient (C_l) for flow past a cylinder with established results from previous studies. The time evolution of these coefficients is shown in Fig. 5.3, demonstrating the periodic nature of vortex shedding, a key characteristic of this type of flow. Additionally, a quantitative comparison of the time-averaged drag coefficient and the amplitude of the lift coefficient is presented in Table 5.1. Our results are consistent with values reported in

the literature, with only minor discrepancies. The slight overestimation of C_l is likely due to the size of the computational domain in the y-direction, which may introduce minor numerical blockage effects.

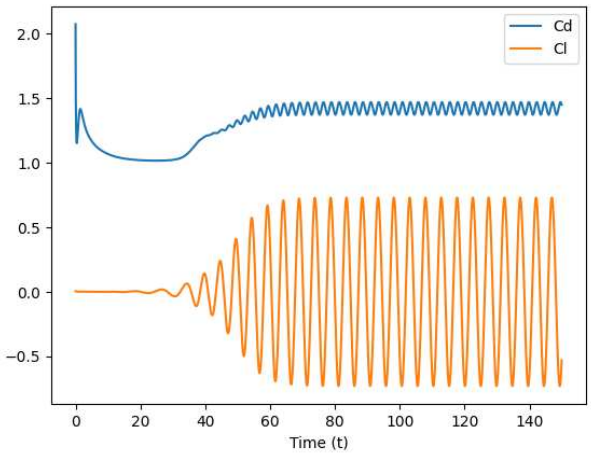


Figure 5.3: Drag coefficient C_d and lift coefficient C_l of flow past a cylinder.

Cylinder	C_d average	C_l amplitude
Etienne and Pelletier [2015]	1.36	0.67
Li et al. [2005]	1.34	0.69
Liu et al. [1998]	1.31	0.69
Present result	1.32	0.73

Table 5.1: Comparison of C_d average and C_l amplitude

5.2.2 . Flow around random shapes

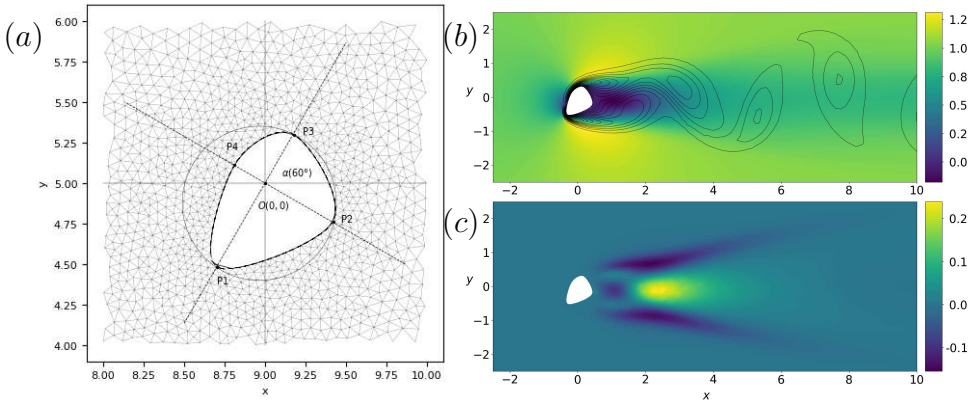


Figure 5.4: Flow past a bluff body with a randomly generated shape at $Re = 90$. (a) Geometry of the bluff body with $\alpha = 60°$ and unitary diameter of the circumscribed circle around the body. The resulting meanflow and the vorticity isolines $\omega = \nabla \times \mathbf{u}$ are shown in (b), while (c) shows the forcing stress term in a portion of the domain.

As mentioned previously, the dataset can be enriched by including different geometries; thus, alongside with the flow past a cylinder, we include in the dataset flow fields around obstacles of random shapes. In principle, we should determine the critical Reynolds number (Re_c) at which the flow around each of the random geometries under analysis develops into unstable baseflows. This would require an extensive campaign of simulations and a comprehensive stability analysis, along the footprints of the recent works by Chiarini et al. [2021, 2022]. Therefore, we adopt a more pragmatic approach by retaining the geometries that in the range $50 \leq Re \leq 150$ develop unsteady flows.

Regarding the geometries of the obstacles, their shape is defined by a set of splines connected at the location of 4 control points, assuring a $C1$ continuity between them. The control points (P1 to P4 in Fig. 5.4a) are located along two orthogonal axes rotated of an angle α with respect to the Cartesian coordinates system of reference. Their distance from the origin $O(0, 0)$ is defined in the range $|s| \in [0.1, 0.6]$, with a discrete step size of $\Delta s = 0.1$. An additional degree of freedom is added for controlling the angle $\alpha \in [0°, 90°]$, with a discrete step size of $\Delta\alpha = 30°$.

Finally, the random shapes generation script prevents the repetition of the same set of degrees of freedom defining the shape such that each geometry appears only once in the final dataset. Note that the center of the circumscribed circle of each random shape does not align necessarily with the center of the coordinate system. This is not an accidental feature: in the context of

NN training, this strategy corresponds to introducing some positional noise in the training dataset, thus promoting a more robust and prone-to-generalize learning of the underlying data. Such variability in the training data further contributes to reduce the risk of overfitting and enhances the NN ability to generalize its predictions to new, unseen geometries.

5.3 . Results

The present section is dedicated to the presentation of the results. We introduce a proof-of-concept of the approach in Sec. 5.3.2; in this first analysis, we train the GNN with a dataset composed of data obtained from simulations of the flow past a cylinder at different Reynolds numbers. The aim is to verify the robustness of the approach with respect to unseen conditions, using the benchmark cases in Sec. 5.3.1. In Sec. 5.3.2, we focus on the appropriate choice of the dataset and how the selection of training data can impact the generalization capabilities of the GNN. In Sec. 5.3.3, we include in the dataset flow fields from simulations of wakes past randomly generated bluff bodies (details of shape generation in Sec. 5.2.2); the effects of dataset augmentation and dataset expansion are discussed in Sec. 5.3.3 and Sec. 5.3.3, respectively, while data selection by active learning criteria is discussed in Sec. 5.3.3. A quantitative comparative analysis of these training approaches is summarized in Sec. 5.3.5; in the following, we introduce the 4 reference cases used for all the comparisons discussed in this section.

5.3.1 . Test Cases

We introduce 4 benchmarks to assess the performance of the GNN model. The test cases, labelled from Case 1 to Case 4, are designed to evaluate specific aspects of the GNN's prediction. We cover a broad spectrum of scenarios, including variability in Reynolds number and geometry, changes in the position with respect of the inlet and number of the obstacles.

1. Case 1: in this scenario, we consider the flow past a cylinder, where the Reynolds number is increased to $Re = 200$. This exceeds the training dataset interval, ranging in $50 \leq Re \leq 150$, and it is used to test the model's capability to extrapolate beyond the training data.

2. Case 2: here, we introduce as a test case the flow past a bluff body of random shape not included in the training dataset at $Re = 120$, in order to assess the model generalization capability to unseen shapes.

3. Case 3: this case involves data from simulation of a flow past a bluff body of random shape, not present in the training dataset, at $Re = 100$. The obstacle is positioned downstream of the reference position used in the dataset.

4. Case 4: this test considers a two side-by-side cylinders configuration; two bluff body obstacles are present in the flow, in contrast to the single obstacle used for the training. The Reynolds number is $Re = 90$ and the aim is to test geometries inducing different dynamics, here the one stemming from multiple interacting bluff bodies.

Fig. 5.5 shows the streamwise component of the meanflow velocity $\overline{\mathbf{u}}$, the isolines of the vorticity $\omega = \nabla \times \mathbf{u}$ (left column) and streamwise component of the forcing stress $\mathbf{f}$ (right column) computed using numerical simulations for the 4 benchmarks Case 1-4.

Two metrics will be used to evaluate the performance of the GNN. The first one will be a comparison between the ground truth $\mathbf{f}$ and the GNN prediction for each of the cases, considered as the relative error ε, based on the $L2$-norm, defined as

$$\varepsilon = \frac{||\mathbf{f} - \hat{\mathbf{f}}||}{||\mathbf{f}||} = \frac{\left[\int_\Omega (\mathbf{f} - \hat{\mathbf{f}})^2 d\Omega\right]^{1/2}}{\left(\int_\Omega \mathbf{f}^2 d\Omega\right)^{1/2}}. \tag{5.3}$$

The second metric employed arises from the necessity to assess the accuracy on the meanflow reconstruction $\hat{\overline{\mathbf{u}}}$ from the GNN prediction with respect to the ground truth meanflow $\mathbf{u}$. It is defined as

$$\delta = \frac{||\mathbf{u} - \hat{\overline{\mathbf{u}}}||}{||\mathbf{u}||} = \frac{\left[\int_\Omega (\mathbf{u} - \hat{\overline{\mathbf{u}}})^2 d\Omega\right]^{1/2}}{\left(\int_\Omega \mathbf{u}^2 d\Omega\right)^{1/2}}. \tag{5.4}$$

In the previous equations, Ω is the computational domain (Sec. 5.2.1), $\hat{\mathbf{f}}$ is the GNN prediction and $\mathbf{f}$ is the closure term of the RANS equations coming from the DNS. Plugging $\hat{\mathbf{f}}$ in Eq. 2.8b and solving the inverse problem, we can obtain $\hat{\overline{\mathbf{u}}}$, a reconstruction of the meanflow based on the GNN prediction.

5.3.2 . Proof-of-concept training: flow past a cylinder flow

Here, we consider the baseline results that in the following we will indicate as proof-of-concept (PC). The training dataset contains data obtained from DNS simulations of the flow past a cylinder; it is composed by 11 pairs of meanflows $\overline{\mathbf{u}}$ (input) and Reynolds forcing stress $\mathbf{f}$ (target), in the interval $50 \leq Re \leq 150$, with a stride of $\Delta Re = 10$. As previously mentioned, all the cases in the dataset exhibit a von Karman street instability in the wake of the bluff body.

Fig. 5.6 shows the training and validation loss curves for this training approach. The training loss behavior underlines the model's ability to capture the dynamics of the fluid flows and to learn the patterns in the training dataset. However, a significant gap can be observed between the training and validation loss curves. This discrepancy suggests a potential problem of overfitting to the training data. This problem is not unexpected, considering that the

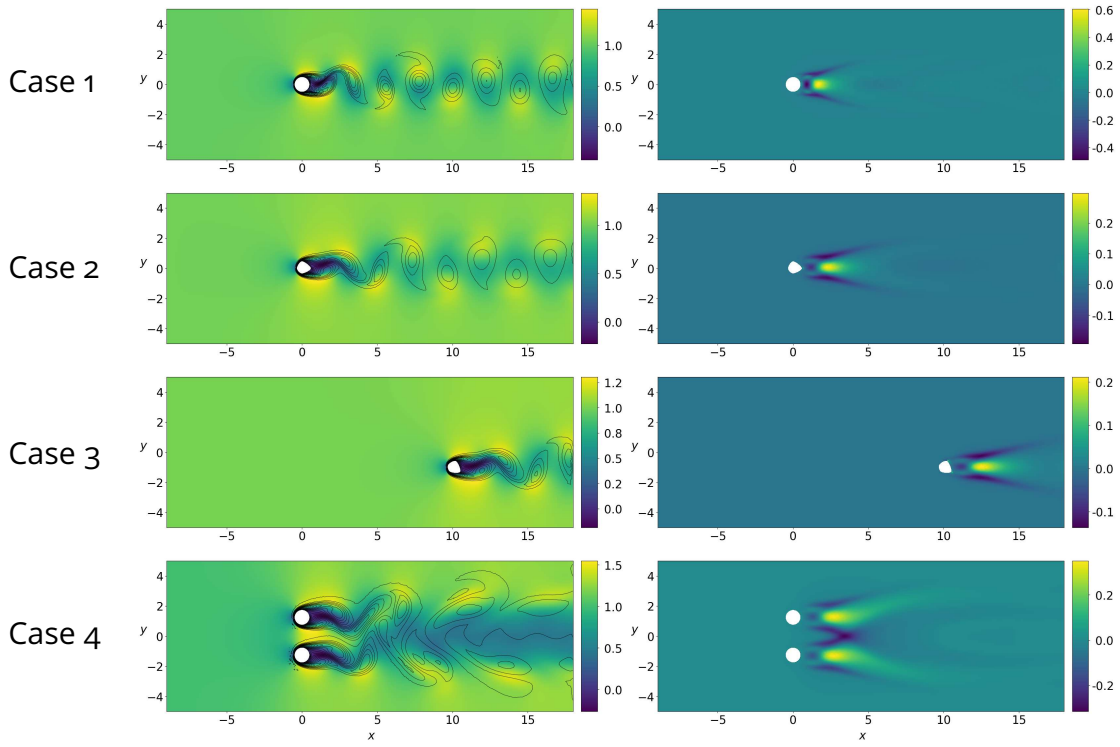


Figure 5.5: Test cases used as a benchmark of the learning strategies discussed in Sec. 5.3. From the top raw to the bottom one: Case 1, cylinder flow at $Re = 200$; Case 2, flow past a random-shaped bluff body at $Re = 120$; Case 3, flow past a random-shaped bluff body shifted in the computational domain at $Re = 100$; Case 4, flow past a two side-by-side cylinders configuration at $Re = 90$. For each of the cases, the left column shows the stream-wise component of the meanflow $\overline{\mathbf{u}}$, along with the vorticity isolines $\omega = \nabla \times \mathbf{u}$, while the right column show the stream-wise component of the forcing stress $\mathbf{f}$.

training dataset is composed exclusively of data obtained from simulations past the same geometry. Thus, when the GNN model is tested with unseen shapes or fluid dynamic conditions, the performance drops.

Regarding the test cases, in Case 1 (Fig. 5.7a), the GNN prediction shows good accuracy in reproducing the fluid structures at higher Reynolds number $Re = 200$, with $\varepsilon = 0.1153$. Fig. 5.7(b) shows Case 2, where the difference between the GNN prediction and the DNS data is primarily visible in the near wake region. This region is crucial for the development of the unstable dynamics as it corresponds to the so-called wavemaker region [Giannetti and Luchini, 2007]; for cases as such of non-standard geometrical shape, the flow vortex shedding behavior can be quite different from the one observed

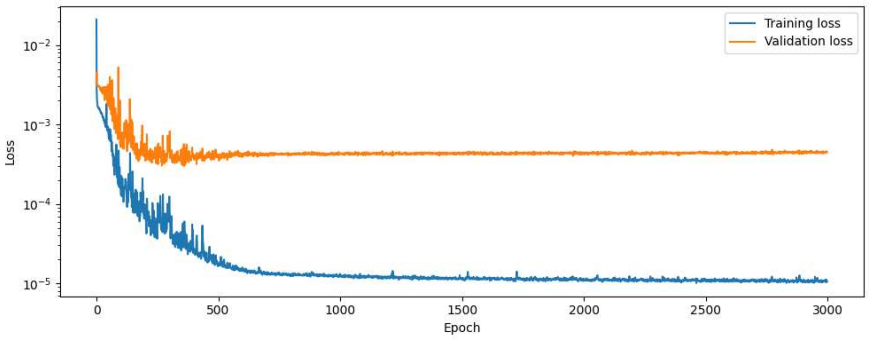


Figure 5.6: Curves for the training and validation loss for the proof-of-concept baseline, trained until 3000 epochs. The training dataset is composed by 11 meanflow-forcing pairs stacked at different Reynolds numbers in the range $50 \leq Re \leq 150$ with $\Delta Re = 10$. The validation set is formed by the test cases presented in Sec. 5.3.1.

in standard cylinder cases. The ε norm in this case is $\varepsilon = 0.2995$. Case 3 in Fig. 5.7(c) tests the GNN generalization capabilities when the bluff body changes; the main discrepancies are again observed in the near wake of the bluff body. The ε norm in this case is $\varepsilon = 0.2084$. Finally, Case 4 in Fig. 5.7(d) is particularly challenging as it tests generalization capabilities in presence of multiple bluff bodies. The GNN model's predictions capture the major features of the forcing stress of each of the cylinder. However, discrepancies are mainly observed in the area between the two cylinders and extending in the far wake region, where interactions between the Reynolds stress fields of the two cylinders occur. The ε norm in this case is $\varepsilon = 0.8704$, significantly higher compared to the previous cases.

In all the cases discussed so far, the presence of some numerical errors in all the predictions is noted, but these are primarily attributed to statistical noise or inherent errors typical of neural network models. Neural networks indeed, by their nature, include elements of statistical uncertainty due to factors such as the stochastic nature of their training algorithms (e.g., random initialization of weights, batch selection during training), and the approximate nature of the model that represents the underlying physics. Beside this aspect, this preliminary analysis based on the benchmark cases suggests that the GNN model performance could be further improved with a training set including a larger variety of bluff body shapes. Therefore, in what follows, we study the effects of the data augmentation by including in the training dataset random shaped bluff bodies with the aim of enhancing the model's robustness and generalization capabilities.

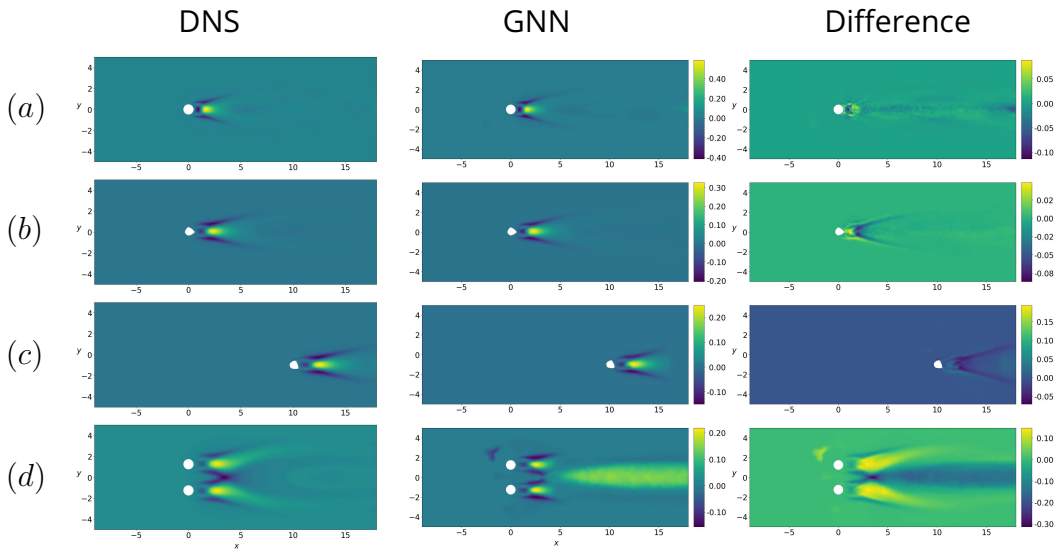


Figure 5.7: Stream-wise component comparison of the Reynolds stress tensor. The GNN's model is trained with a dataset composed by 11 cases of cylinder shape bluff bodies, ranging in $50 \leq Re \leq 150$, $\Delta Re = 10$. (a) Case 1, $Re = 200$, cylinder bluff body shape; (b) Case 2, $Re = 120$, random shape bluff body; (c) Case 3, $Re = 100$, random shape shifted bluff body; (d) Case 4, $Re = 90$, flow past two side-by-side cylinders.

5.3.3 . Data augmentation and active learning: fluid flows past random geometries

In the following, we study the effects of data amount and quality on the training process and generalization properties of the GNN models. This study is structured into distinct steps.

1. **Data augmentation (DA)**: this first approach involves dataset augmentation, meaning that we incorporate bluff bodies of random shapes cases into the existing dataset, while maintaining the same dataset size used in the PC case, *i.e.* 11 cases. This method is aimed at diversifying the range of geometries used during the training of the model, without increasing the data volume.

2. **Dataset expansion**: in this second approach, we expand the dataset by including 33 cases of bluff bodies of random shapes. We assess the quality of the used data in terms of sensitivity of the model to specific configurations by performing a k-fold validation.

3. **Active learning (AL) data selection**: in this last approach, the training set is built by adding progressively data chosen by similarity criterion. The goal is to develop a surrogate model that can generalize at its best to unseen cases based on a given dataset.

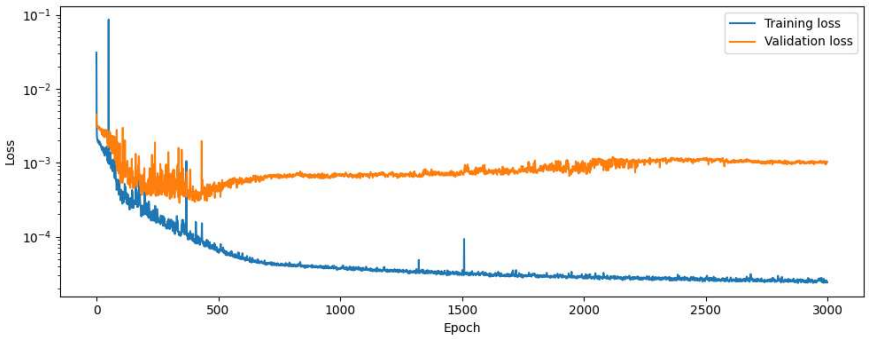


Figure 5.8: Curves for the training and validation loss for when DA is performed by introducing 11 cases of flows past random geometries in the dataset. The Reynolds number varies in the range $50 \leq Re \leq 150$ with $\Delta Re = 10$. The training runs until 3000 epochs. The validation set is formed by the test cases presented in Sec. 5.3.1.

1 - Data augmentation with random shapes

In this phase of the study, we employ a stratified random sampling criterion to select shapes among the randomly generated configurations detailed in Sec. 5.2.2. This latter involves the generation of the entire set of shapes for each Reynolds number in the interval $50 \leq Re \leq 150$, $\Delta Re = 10$ and randomly select 1 shape for each value of the Reynolds number, for a total of 11 shapes that compose the new training dataset. The chosen geometries are unique. On the selected shapes, we perform DNS simulations in order to obtain the meanflow (input) and the forcing stress (target) used to train the GNN.

Fig. 5.8 shows the training and validation loss curves for this second training approach. The training loss demonstrates a consistent downward trend, highlighting the learning effectiveness of the GNN model. Notably, while there remains a significant gap between the training and validation loss curves, this gap is less pronounced when compared to the initial training approach (Fig. 5.6). This reduced gap is indicative of diminished overfitting and suggests that the introduction of a wider variety of shapes and flow conditions into the training dataset leads to improved GNN model's generalization capabilities.

Regarding the GNN predictions on test cases, Case 1 (Fig. 5.9a) shows larger discrepancies in the forcing prediction in the neighborhood of the cylinder compared to the training approach discussed in Sec. 5.3.2, $\varepsilon = 0.5033$. An interpretation for this result is given by the nature of the PC training dataset that solely relies on cylindrical geometries; thus, the resulting GNN model is specialized in the prediction of the cylinder flows. In contrast, the current training strategy involves flows past random shapes: from one hand, this choice broads the GNN's flexibility to capture diverse flow conditions by enhancing generalization capabilities; on the other hand, it simultaneously decreases the

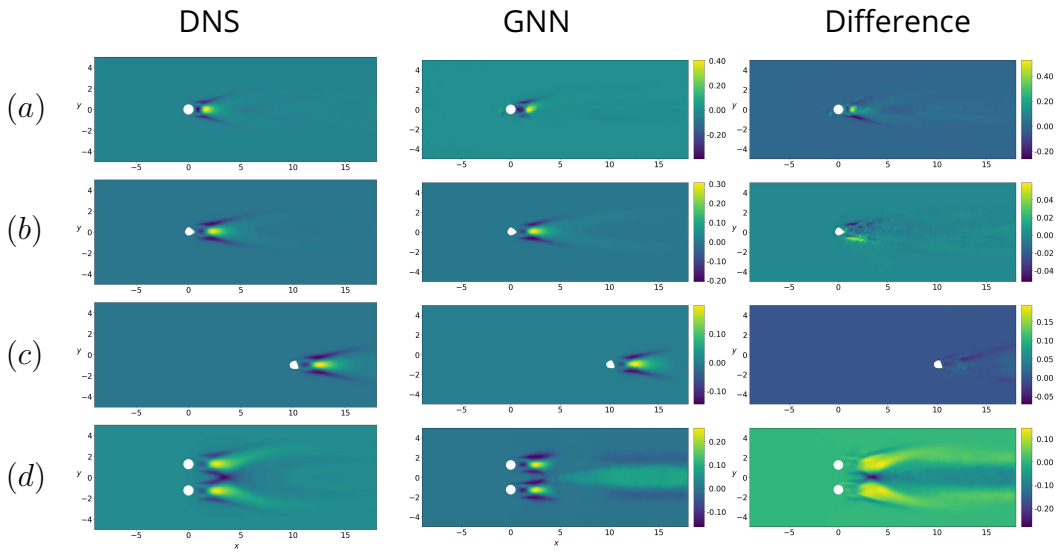


Figure 5.9: Stream-wise component comparison of the Reynolds stress tensor. The GNN's model is trained with a dataset composed by 11 cases of random shaped bluff bodies, ranging in the interval $50 \leq Re \leq 150$, $\Delta Re = 10$. (a) Case 1, $Re = 200$, cylinder bluff body shape; (b) Case 2, $Re = 120$, random shape bluff body; (c) Case 3, $Re = 100$, random shape shifted bluff body; (d) Case 4, $Re = 90$, flow past two side-by-side cylinders.

prediction accuracy for cylinder shaped cases.

Case 2 (Fig. 5.9b) and Case 3 (Fig. 5.9c) show mainly numerical and statistical noise in the difference fields while the main features of the Reynolds stress tensor are well reproduced in the prediction. Contrary to what is observed in Case 1, this training approach leads to an efficient GNN model in predicting the features of the flows past random shaped bluff bodies. Error norm is comparable for the two test cases, namely $\varepsilon = 0.2063$ for Case 2 and $\varepsilon = 0.1999$ for Case 3. Finally, Case 4 is characterized by inaccuracies primarily located in the wake region similar to the one already observed in the PC case (for a reference, the reader can compare Fig. 5.7d and Fig. 5.9d). However, it's important to note that these errors are quantitatively less significant to those observed in the first training approach, with $\varepsilon = 0.6312$.

2 - Dataset Expansion

A common technique to improve generalization in neural network models is to enlarge the volume of data the model is trained with. This approach reduces the risk for the NN model to overfit to specific conditions observed in a small dataset and is more likely to capture the underlying phenomena. Therefore, we study the effect of tripling the amount of data in the training dataset. With the same stratified random sampling approach described in the previ-

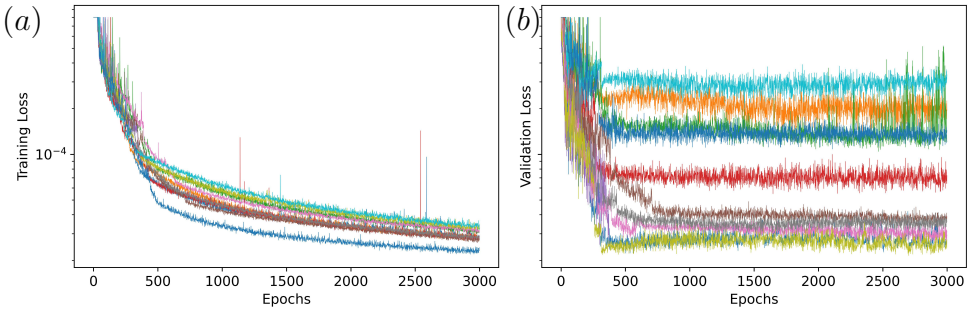


Figure 5.10: Training and validation loss curves for models trained using 10 k-fold up to 3000 epochs. (a) shows the training curves, (b) the corresponding validation curves for each fold.

ous paragraph, we select 3 cases for each of the 11 Reynolds numbers chosen in the interval of reference, resulting in a 33 cases training dataset. In order to analyze the sensitivity of the GNN model's generalization capabilities with respect to the training dataset, we employ a k-fold validation test. First, we divide the training dataset into k groups and train k different models. For each of the k models, we use $k-1$ groups as the training dataset and the k-th remaining group as the validation dataset. Although this approach is not feasible in practical applications, it is very informative for assessing preliminarily the impact of the quality of data on the final prediction.

Fig. 5.10(a) shows the training curves of the 10 models, one for each of the k folds, while Fig. 5.10(b) displays their corresponding validation curves. The training curves are rather close, independently of the chosen dataset, thus suggesting a rather consistent behavior of the chosen GNN architecture. However, the validation loss varies significantly. This variability can be attributed to the distribution of the training dataset. Inconsistent or unbalanced data distributions may lead the model to being trained on subsets of data that do not adequately represent the overall dataset, thus impacting negatively on the validation results. To address this challenge and identify the most effective dataset, we introduce AL in the following.

3 - Active Learning data selection

AL allows to dynamically adjust the dataset, targeting the data that contribute most significantly to the model's generalization capability. We exploit a similarity criterion from the GNN perspective, following the study by Charpiat et al. [2019]. The key idea is to avoid including in the training dataset cases that do not lead to any significative improvement of the model from the generalization viewpoint; instead, in order to induce diversity in the GNN model,

we need to select cases that are as most as different from each other from the GNN perspective. We can achieve this goal by comparing, for each of the available data pairs, the vector gradient of the cost function with respect to the θ learnable parameters of the GNN by means of scalar products. As a comparison metric, we employ the cosine similarity angular distance

$$\cos(\beta) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|}, \tag{5.5}$$

where $\mathbf{a} \in \mathbb{R}^m$ and $\mathbf{b} \in \mathbb{R}^m$ are two generic m-dimensional vectors and β is the angle between them. The basic idea is that each gradient vector steers the neural network's state to a specific direction within the solution space. Therefore, including multiple data points whose gradients point in the same direction might be redundant. Instead, our aim is to select and include in the training dataset as many different directions as possible, in order to explore more extensively the solution space (see also 5.3.4).

The selection process stops when a predefined similarity threshold is reached, such that only the pairs exhibiting a similarity below this threshold are added to the dataset. Conversely, cases that show a similarity value exceeding the threshold are discarded. However, the similarity threshold does not have a direct, interpretable meaning and needs to be tuned as an input parameter of the process, according to the required performances. In general, a lower similarity threshold means that more cases will be included in the training dataset. While this can enhance models behavior to diverse scenarios, it may also lead to a more complex and time-consuming training process. On the other hand, a too restrictive similarity threshold might exclude potentially valuable training data. Therefore, an optimal similarity threshold is chosen as a trade-off between the training computational costs and the model generalization performances.

In our study, we explore three different similarity threshold values, $\cos(\beta) \in [0.7, 0.8, 0.9]$. This approach allows to observe the impact of varying levels of data inclusion on the model's training and performance. Results shown in Fig. 5.12 and Fig. 5.11 are obtained with a similarity threshold of 0.8, which results in only 6 selected random bluff body cases for the training dataset; the results for the other threshold values are detailed in Sec. 5.3.5. Fig. 5.11 reports the training and validation loss curves: the reduced disparity observed between the two curves – compared to the results from the previous learning approaches – underscores a notable decrease in the issue of overfitting. The training is initiated with a single case in the training dataset, specifically the cylinder case at $Re = 120$.

Once the convergence of the vector gradients is reached (Sec. 5.3.4), a new case in the training set is selected; since this process is a seek-and-include algorithm that enlarges the dataset at each step, two different approaches can

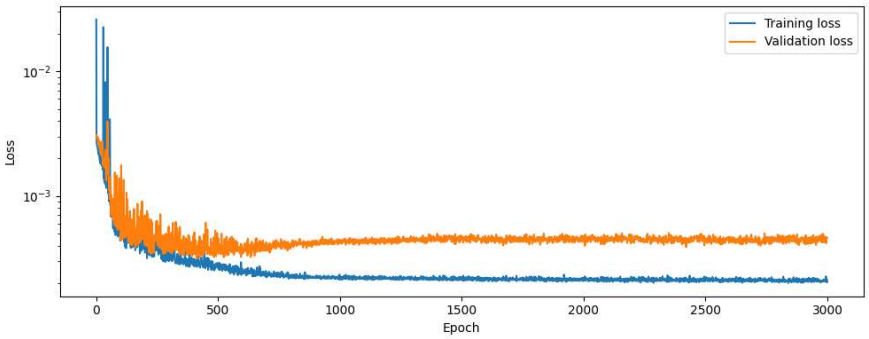


Figure 5.11: Training and validation loss curves are shown when AL with similarity threshold 0.8 is performed until 3000 epochs. The training dataset is formed by 6 random shaped bluff bodies. Their Reynolds number varies in the interval $50 \leq Re \leq 150$. The validation set is formed by the test cases presented in Sec. 5.3.1.

be applied. The first strategy is inspired by the curriculum learning framework [Bengio et al., 2009], where each selected case is added to the ongoing training. Thus, the model is progressively updated. A second approach, applied in this work, consists in reinitializing the GNN weights every time the dataset is enlarged with a new pair. The rationale behind this choice is based on the dynamics of the solution space, that changes when new data are added to the training set; in this sense, there is no guarantee that the new minima will be "closer" to the previous ones than to the initialization point of the GNN. Nonetheless, we observed that for the analyzed flow cases the two training strategies lead to negligible differences in the final results.

Considering the benchmarks for the model assessment, Case 1 leads to $\varepsilon = 0.5507$ between the GNN predictions of the Reynolds stress tensor and the DNS ones. The error is $\varepsilon = 0.1518$ for the random shape bluff body case (Case 2), $\varepsilon = 0.3595$ for the downstream shifted random shape bluff body (Case 3) and $\varepsilon = 0.5243$ for the two side-to-side cylinder configuration (Case 4). The results respect the symmetry of the solutions with respect of the y axis and only minor inconsistencies can be observed in the far wake; we note that the errors are concentrated in the region immediately downstream of the bluff body, but we interpret these errors as numerical or stochastic in nature rather than being prediction inaccuracies of the fluid structures.

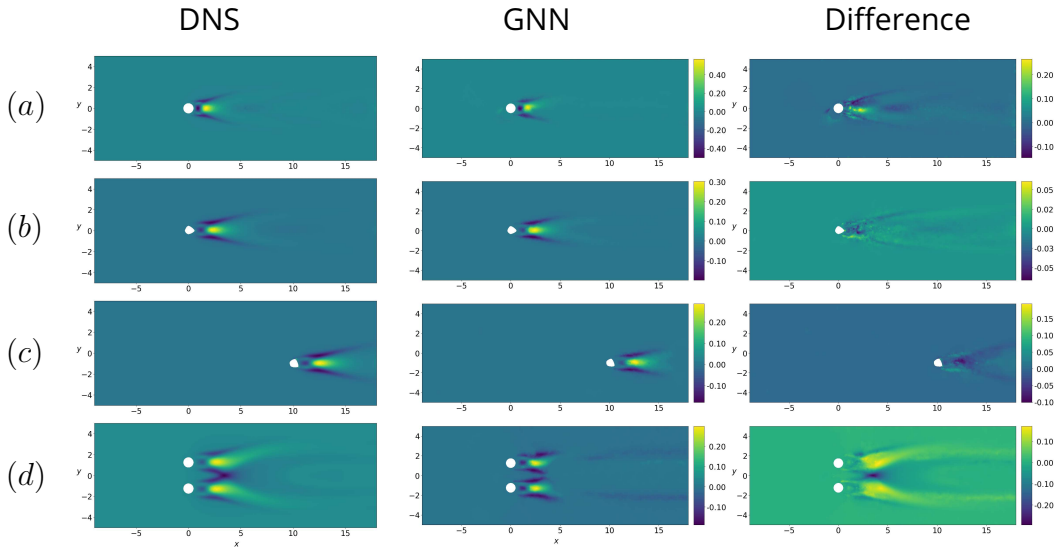


Figure 5.12: Stream-wise component comparison of the Reynolds stress tensor. The GNN's model is trained with a dataset composed by 6 cases of bluff bodies of random shape selected with the AL approach, ranging in the interval $50 \leq Re \leq 150$, $\Delta Re = 10$. (a) Case 1, flow past a cylinder at $Re = 200$; (b) Case 2, flow past a random shaped bluff body at $Re = 120$; (c) Case 3, random shaped bluff body at $Re = 100$, shifted downstream; (d) Case 4, flow past two side-by-side cylinders at $Re = 90$.

5.3.4 . Similarity criteria algorithm details

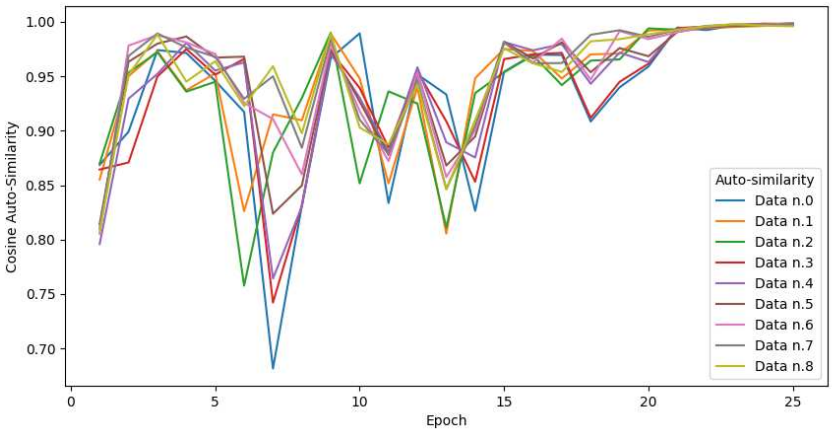


Figure 5.13: Visualization of gradient auto-similarity convergence over multiple training epochs using MAE for 9 distinct cases within the training data set.

The similarity criteria algorithm designed to compare different data from the neural network perspective is based on the analysis of the vector gradients of a metric function (Eq. 5.7) with respect to the θ parameters of the neural network. In particular, the similarity comparison between two generic m-dimensional vectors $\mathbf{a} \in \mathbb{R}^m$ and $\mathbf{b} \in \mathbb{R}^m$ is computed using the cosine similarity, defined as

$$\cos(\beta) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|}, \tag{5.6}$$

where β is the angle between the two vectors. In this context the metric we use is the Mean Absolute Error (MAE), a piecewise linear function defined as:

$$MAE = \sum_{i=1}^{n_i} |x_i - y_i|, \tag{5.7}$$

in which x_i is the NN prediction on the node i, y_i the ground truth and n_i the number of nodes.

The choice of MAE is crucial for our analysis. In scenarios where the Mean Squared Error (MSE) is used, we empirically observe marked oscillations in the direction of the auto-similarity of the vector gradient throughout successive training epochs. Conversely, when employing MAE, the auto-similarity of the vector gradient tends to approach unity, suggesting a stable direction in the solution space, for the data under analysis. A visual representation of the consistent convergence of the vector gradient auto-similarity as function of the epochs is shown in Fig. 5.13, for a training process involving 9 cases in

the training dataset. Training begins with a specific initial dataset; when every data point in the training dataset achieves an auto-similarity convergence exceeding a predefined threshold of 0.99, the training is stopped and we can assume that the vector gradient's direction for each instance in the training dataset has stabilized.

The following step is to assess the similarity between cases within the training dataset and those outside it. This aims at identifying the most diverse cases among those not included in the training set, which will then be added to the training dataset to enhance diversity. Firstly, the GNN runs for additional 10 epochs to obtain the vector gradients of each out-of-training dataset instance. Then, a similarity matrix is computed by cross calculating the similarity between each in-training instance and each out-of-training instance. The case that shows the lowest similarity score is also the most diverse one and enables to promote diversification in the training dataset based on available data. Note that the values are normalized using a z-score value

$$z = \frac{S - \mu}{\sigma}, \tag{5.8}$$

where S represents the similarity score for a specific data point, μ is the mean of all similarity scores, and σ their standard deviation. In summary, the approach outlined here serves as a robust method for comparing and evaluating the similarities in data behavior leveraging the neural network model, thereby enhancing the efficacy of the training process.

5.3.5 . Quantitative comparison

In conclusion, we summarize the results in Tab. 5.2. A direct comparison between the first two training approaches (PC and DA) reveals that enriching the training dataset with random geometries enhances the generalization capabilities of the GNN. This behavior is not unexpected, although an exception is observed when extrapolating at higher Reynolds numbers (Case 1), where we observe that the PC approach outperforms DA. As already discussed, this behavior can be understood by observing that the GNN model trained following the approach PC is specialized in predicting the flow past a cylinder as it is trained on this specific geometry.
On the other hand, the AL training approach allows to obtain GNN models demonstrating overall superior performance in terms of generalization capabilities as compared to PC and DA approaches, in particular for Case 2 and Case 4, while in Case 3 performance are essentially comparable. The mean error decreases in all the 4 test cases in the AL approach, indicating an improvement in the global performance of the model. An important aspect regarding the active selection of data for the training is the amount of data. Good performances are achieved with only 6 pairs when a threshold 0.8, although it is also possible to observe a strong variation in the total number of

Cases	PC		DA		AL					
					0.7		0.8		0.9	
	ε	δ	ε	δ	ε	δ	ε	δ	ε	δ
Case 1	**0.1153**	*0.0118*	0.5033	0.1987	0.8671	0.1859	0.5507	0.1165	0.4132	0.1162
Case 2	0.2995	0.0280	0.2063	0.0160	0.1256	*0.0079*	0.1518	0.0166	**0.1124**	0.0092
Case 3	0.2084	0.0131	**0.1999**	0.0090	0.3058	0.0107	0.3595	0.0183	0.2123	*0.0082*
Case 4	0.8707	0.2683	0.6312	0.1701	0.6751	0.1930	**0.5243**	*0.1565*	0.5719	0.1653
Training data	11		11		3		6		19	

Table 5.2: Comparison on the 4 cases defined as benchmark for the three different training approaches: the proof-of-concept (PC) training (Sec. 5.3.2), the data augmentation (DA) training (Sec. 5.3.3), and the active learning (AL) strategy (Sec. 5.3.3). For the latter, we consider three similarity threshold values $\cos(\beta) \in [0.7, 0.8, 0.9]$. The chosen metric ε and δ are defined respectively in Eq. 5.3 and Eq. 5.4. In the last row, the number of pairs used during the training process is reported.

pairs used for the training as a function of the chosen threshold. It is crucial to observe, however, that for all the AL cases the introduction of a selection criterion guarantees that the chosen data points lead to good performance in terms of generalization.

When comparing our work with existing literature, the notable aspect is that our approach achieves comparable accuracy with significantly fewer training cases. For instance, in Chen et al. [2021] the authors utilize 2000 cases in their training dataset for steady-state incompressible flow around a cylinder at $Re = 10$. In contrast, we use only 19 cases for the largest dataset used and still manage to generalize to different Reynolds numbers and bluff body positions, with comparable accuracy results. In Lee and You [2019], $500k$ cases are used for the training dataset, although an unsteady flow is predicted using CNN. It is worth noting that the use of a GNN architecture enables to generalize on different geometries and Re numbers, an aspect that is not addressed in Lee and You [2019]. Finally, in Thuerey et al. [2020], a GNN is employed for predicting time–averaged steady flow. The dataset consists of 12800 data points while the GNN model has a complexity of over $30M$ parameters, prohibitive for most practical applications. Our GNN architecture, on the contrary, can count up to approximately $900k$ parameters. In conclusion, we believe that the combination of GNN models and active learning makes our method more accessible and practical for broader applications thanks to the parsimony of the data requirements.

5.4 . Discussion

The application of machine learning techniques in fluid mechanics is often characterized by shortcomings such as overfitting, lack of robustness of the prediction with respect to unseen cases and data-hungriness. In this study, we proposed a novel approach that combines a neural architecture based on Graph Neural Network (GNN), numerical solvers based on Finite Element Method (FEM) and an active learning procedure in order to tackle some of these limitations. Here, we consider a data-assimilation schemes that does not rely on an optimization process and use as baseline equations the Reynolds Averaged Navier-Stokes (RANS) equations. GNN models are trained as a surrogate to predict the forcing/closure term, obtained as an output of the supervised learning, while a given mean flow serves as input. The GNN architecture is particularly suitable in this study due to its adaptability to unstructured meshes and its generalization capability, as compared to other literature approaches. Moreover, this architecture allows frugal training within the low-data limit, as compare to alternative, more expensive in terms of required data, architectures.

A two-fold interface between FEM and GNN environment has been developed to transform a FEM vector field into a numerical tensor that can be handled by a NN structure and vice versa, preserving critical information throughout the process. As a test-bed, we focussed on two dimensional, incompressible flows past obstacles at low Reynolds numbers, namely in the range $50 \leq Re \leq 150$. At these regimes, the presence of obstacles triggers instabilities developing in unsteady flows. We started by studying a cylindrical geometry in the range $50 \leq Re \leq 150$ as initial benchmark, in order to assess the extent to which a model based on this training dataset can be used also for unseen cases. Not surprisingly, we found good performance at unseen Reynolds number for the cylinder case. On the other hand, when the flow around the bluff bodies of random geometry is considered, it is observed lack of accuracy in the prediction and overfitting.
In order to tackle these limitations, we explored the impact of the training data on the generalization capabilities of the GNN in terms of quantity and quality of data. First, we considered an extended dataset. Our results indicate that the quality and volume of data notably affect the spectrum of unseen cases on which the model can generalize to. Particularly, the inclusion of diverse fluid flow conditions into the training dataset improves the overall generalization capabilities for the vast majority of cases. Finally, we introduced an active learning data selection criterion based on the analysis of gradient similarity, with the aim of building a dataset extending the distribution of the data. At the best of authors knowledge, this is one of the first applications in the community of fluid mechanics where a systematic selection of the data is performed

addressing the generalization of the NN model prediction to unseen cases by maximizing the quality of the predictions and at same time minimizing the amount of data used in the training set. The results clearly indicate the possibility of improving the performance of the model, also in terms of generalization, while keeping a rather small amount of data in the training set. The criterion is especially relevant in the contexts where computational resources for training surrogate models are limited, and a trade-off between accuracy of the predictions and training computational cost is sought. It is stressed that the datasets used during the training process are relatively small as compared to other approaches appeared in ML literature, as the most expensive one consists of less than 20 pairs of snapshots. These datasets are selected through a criterion that minimizes the number of data points required and is robust enough to be applied to larger datasets to efficiently reduce them while maintaining essential information. In terms of future studies, this project has many

potential directions and developments that can be pursued. Among them, one possible development is the implementation of a sensitivity analysis of the GNN predictions by leveraging the gradients of the model. Specifically, this involves analyzing the sensitivity of GNN predictions with respect to variations in the input parameters, such as the mean flow. This could be followed by an adaptive training procedure where a weight mask is applied to the loss function to concentrate the training effort on the most sensitive regions of the flow. For example, the GNN could initially predict a sensitivity map of the input mean flow, highlighting the zones with the highest influence on the predicted field. The loss function could then be adapted for each specific case to focus the training more accurately on these sensitive areas, leading to more efficient and localized learning.

A similar approach could be adopted using uncertainty quantification. By quantifying the uncertainty in the GNN's predictions, regions with high uncertainty could be identified, and the training could be adaptively focused on these areas to reduce the model's uncertainty and increase robustness. This would involve dynamically adjusting the loss function to prioritize regions with higher uncertainty, thereby guiding the model to learn where it is most needed.

Another promising direction is exploring different neural network architectures, such as Recurrant Neural Networks (RNN)s with adaptive update mechanisms. This could involve an adaptive number of iterations to improve efficiency and accuracy depending on the training case [Nastorg, 2024].

Transformers [Vaswani et al., 2017], with their ability to capture long-range dependencies, could be another architecture worth exploring. Unlike GNNs, which excel in capturing local relationships, transformers could help in understanding global interactions between distant points in the flow field. Leveraging a combination of these two approaches could lead to potential benefit in

terms of accuracy and generalization capabilities.

In the next Chapter we will extend our findings by incorporating physical constraint in the learning loop through the adjoint equations associated with the assimilation loop. This integration aims to further refine the predictive performance and generalization capabilities of the GNN leveraging the physics constraints. The final goal of these data-assimilation schemes is to adapt our training approach to cases where solely limited or corrupted measurements of the flow are available, such as those based on sparse probe measurements, noisy or incomplete data.

6 - Part II: Physics-Constrained Graph Neural Network (PhyCo-GNN)

6.1 . Introduction

In recent years, the integration of Machine Learning (ML) algorithms into Computational Fluid Dynamic (CFD) has seen a significant boost, driven by the increasing efficiency of ML models in processing large dataset and their impressive inference and predicting capabilities.
Literature is already disseminated with different, effective ways to combine ML algorithms into CFD, as can be found in the annual review by Brunton et al. [2020b] and in Vinuesa and Brunton [2022]. These applications range from addressing the closure problem of Reynolds-averaged Navier-Stokes (RANS) equations to optimization problems. Notably, Duraisamy et al. [2019] and Beck and Kurz [2021] provide a thorough overview of ML techniques applied specifically to turbulence modeling and RANS equation closure. In Ling and Templeton [2015], authors used classification methods to identify regions of high uncertainty in RANS fluid flow predictions. In Ströfer and Xiao [2021], the authors combined NN with a Spalart-Allmaras turbulence baseline model to enhance fluid flow RANS predictions. Data-assimilation techniques are also been explored to enhance the turbulence models of RANS equations, as can be found in the extensive study by Cato et al. [2023]. However, despite the potential of ML models, one of the persistent challenges is ensuring that the learned solutions adhere to fundamental physical laws. Unconstrained ML models may yield results that violate physical principles, undermining the reliability and interpretability of simulations. To address this problem, an established approach is the use of Physics-Informed Neural Network (PINN)s, which incorporate physical equations as part of the NN training process to maintain physical consistency Cai et al. [2021].

In this Chapter, we propose a novel approach by combining Graph Neural Networks (GNNs) (Sec. 3.7) as ML framework with Reynolds-Averaged Navier-Stokes (RANS) (Sec. 2.3) equations as our physical baseline model. GNNs are particularly suited for CFD problems due to their ability to handle complex geometries, often encountered in fluid flow simulations. They extend traditional neural networks by considering the relationships between data points, making them ideal for capturing the particles interactions in a fluid flow system.
Our primary goal is to develop a hybrid ML-CFD model to accurately reconstruct the mean flow of a fluid dynamics simulation across various application cases. Traditionally, mean flow reconstruction has been tackled using data-

assimilation techniques that combine experimental measurements with CFD models. Foures et al. [2014] proposed a variational data-assimilation method that uses forced RANS equations to reconstruct the mean flow field from partial measurements. This method minimizes the discrepancy between the experimental data and numerical solutions by identifying the optimal forcing term that represents the unknown Reynolds stresses.
More recently, ML techniques have been explored to enhance flow reconstruction. Belbute-Peres et al. [2020] developed a hybrid model that combines GNNs with differentiable fluid dynamics solvers. This approach leverages the efficiency of NNs while maintaining the accuracy of physical solvers. Furthermore, Chen et al. [2021] demonstrated the use of GNNs for predicting laminar flows around arbitrary 2D shapes, showing promising results in terms of accuracy and computational speed compared to traditional solvers.

Specifically, we aim to integrate RANS equations into a GNN training process, leveraging the RANS closure term as an optimization term through the adjoint method (Sec. 4.5). Adjoint method is a powerful mathematical tool used in CFD to compute gradients efficiently, which are essential in a classical optimization process. We use the adjoint method to ensure that the gradients used in the GNN training process are obtained through a deterministic physical model. With this approach, we can train the ML model by guaranteeing physical consistency and leading to improved performance and accuracy with respect to supervised learning or standard methods. We test our approach on different CFD scenarios, showing remarkable improvements in mean flow reconstruction accuracy for different learning tasks as compared to the non physics constrained counterpart.

The physical baseline model for the CFD simulations is detailed in Sec. 2.3.3. The numerical setting used in this study is inherited by the previous study 5, specifically in 5.2.1. The adjoint optimization method can be found in Sec. 4.5.1. Sec. 3.7 describes the ML framework, detailing the custom GNN architecture used in (Sec. 3.7.1), the dataset preprocessing (Sec. 3.8.1) and the training algorithm (Sec. 3.8.2). We continue, then, by presenting our innovative approach to combine these two frameworks in Sec. 6.2. Results, along with the different application cases, are presented in Sec. 6.3.

6.2 . Methodology

This section describes the methodology developed here, combining RANS (Sec. 2.3.3) and the training of a GNN model (Sec. 3.7). The main focus of the approach relies on the use of gradients derived analytically from the RANS equations through the adjoint method (Sec. 4.5) to enhance the learning process of the GNN and ensure physical consistency in its predictions. The com-

plete training process is shown in Fig. 6.1. In the following, Sec. 6.2.2 gives some technical details on the pre-training phase of the GNN model, while Sec. 6.2.3 details the approach adopted to combine the transition between the pre-training phase and the effective training of the GNN.

6.2.1 . The training process

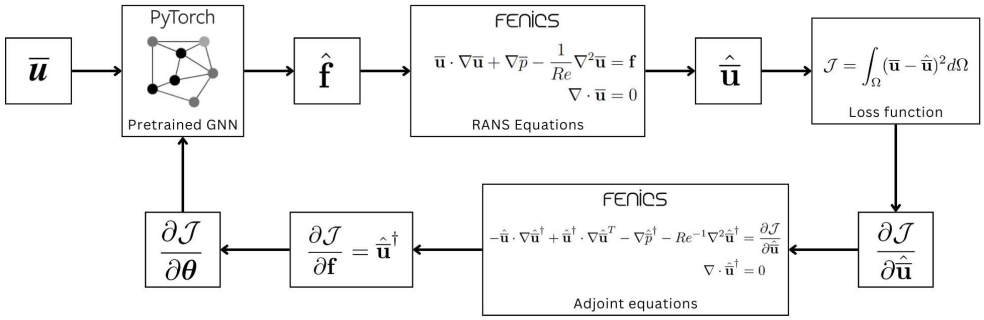


Figure 6.1: End-to-end training loop; $\overline{\mathbf{u}}$ is the GNN's input mean flow; $\hat{\mathbf{f}}$ is the GNN's predicted forcing stress term; θ are the GNN's trainable parameters; $\mathcal{J}(\hat{\overline{\mathbf{u}}})$ is the cost function to minimize.

With reference to Fig. 6.1, the global training process can be ideally divided into two phases, the forward and the backward step. The forward step begins with the input of the mean flow $\overline{\mathbf{u}}$ (and Reynolds number Re) into a pretrained GNN (Sec. 6.2.2), which predicts a forcing stress term $\hat{\mathbf{f}}$. This predicted forcing term is plugged into the direct RANS equations (Eq. 2.8b). By using the Finite Element Method (FEM) approach, handled by the python library `FEniCS` [Alnæs et al., 2015], we solve numerically the RANS inverse problem to obtain a mean flow prediction $\hat{\overline{\mathbf{u}}}$. This result is then compared with the mean flow ground truth $\overline{\mathbf{u}}$ obtained from the DNS to compute a loss function $\mathcal{J}$ that needs to be minimized:

$$\mathcal{J} = \int_{\Omega} (\overline{\mathbf{u}} - \hat{\overline{\mathbf{u}}})^2 d\Omega. \tag{6.1}$$

Eq. 6.1 is computed directly in the FEM environment as an integral over the entire computational domain Ω of the squared difference between the predicted mean flow $\hat{\overline{\mathbf{u}}}$ and its ground truth $\overline{\mathbf{u}}$.
The second phase, the backward step, starts with the requirement to compute the derivative of the loss function $\mathcal{J}$ with respect to the $\boldsymbol{\theta}$ parameters of the GNN. The gradient chain rule for this required term can be mathematically expressed as:

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{J}}{\partial \hat{\overline{\mathbf{u}}}} \cdot \frac{\partial \hat{\overline{\mathbf{u}}}}{\partial \hat{\mathbf{f}}} \cdot \frac{\partial \hat{\mathbf{f}}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{J}}{\partial \hat{\mathbf{f}}} \cdot \frac{\partial \hat{\mathbf{f}}}{\partial \boldsymbol{\theta}}. \tag{6.2}$$

The first term $\frac{\partial \mathcal{J}}{\partial \hat{\mathbf{f}}}$ of the right-hand side is obtained from Eq. 4.20 after solving the adjoint equations (Eq. 4.18b). The second term $\frac{\partial \hat{\mathbf{f}}}{\partial \boldsymbol{\theta}}$ of the right-hand side is indeed the gradient of the GNN's output with respect to the $\boldsymbol{\theta}$ parameters of the GNN, which is available using the automatic differentiation. These two gradients, the analytical one discretized using `FEniCS` and the numerical one obtained by the automatic differentiation included in `PyTorch Geometric` [Fey and Lenssen, 2019] are combined together to complete the chain rule. Finally, these compounded gradients are used to train the GNN.

6.2.2 . On the pre-training step

A crucial step of the algorithm is the GNN model's pre-training phase. This step is necessary to ensure that the GNN's prediction is plausible enough to be plugged into the RANS equations. Indeed, the GNN model's weights and biases are defined using a default initialization [He et al., 2015] and therefore early GNN's predictions are non-physical and cannot be reliably used in the forward step where the forcing is introduced in the RANS equations for the computation of the mean flow (Sec. 6.2.1). Indeed, the solution to the RANS inverse problem may not exist if the initial guess for the forcing term, $\hat{\mathbf{f}}$, is too far from a physical value. The pre-training step helps in stabilizing the GNN's output and overcome this problem, making the forcing stress term $\hat{\mathbf{f}}$ prediction suitable for subsequent integration into the RANS equations. The pre-trained model is obtained via a pure supervised learning of the mapping between the mean flow $\overline{\mathbf{u}}$ (and Reynolds number Re) used as input and the forcing stress term $\mathbf{f}$ as target, both coming from DNS. The loss function used in this phase is the Mean Squared Error (MSE) loss $\mathcal{M}$, reading as

$$\mathcal{M} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{f}_i - \hat{\mathbf{f}}_i)^2, \tag{6.3}$$

where n is the number of nodes of the GNN. The number of epochs needed to reach the required closure term accuracy depends on the specific case at hand, and it will be specified for each of the training cases shown in the result section (Sec. 6.3). The closure term accuracy, in this context, refers to the level of precision necessary for the GNN to produce predictions that enable the FEM solver to successfully solve the RANS equations. Throughout the pretraining phase, the GNN predictions are periodically evaluated by solving a test FEM step. If the solver converge and accurately resolve the RANS equations using the GNN predicted closure term, the pretraining phase is considered complete. This ensures that the GNN has learned an accurate and reliable representation of the closure term, making it suitable for advancing to the full training scheme.

6.2.3 . On the loss function

During the pre-training step (Sec. 6.2.2), the GNN model is updated using a loss function designed to align the model's predictions with the available data from DNS. As already stated, this phase can be seen as a warm-up step of the subsequent main training (Sec. 6.2.1). However, when the pre-training ends and the main training begins, a different loss function is adopted, as can be observed by comparing Eq. 6.3 with Eq. 6.1. This change may be detrimental in terms of convergence and destabilize the training process, as the two optimization landscapes can be significantly different. To mitigate this risk, both loss functions are retained during the main training phase and combined through a weight coefficient β as:

$$\mathcal{L} = (1-\beta)\mathcal{M} + \beta\mathcal{J} = (1-\beta)\left(\frac{1}{n}\sum_{i=1}^{n}(\mathbf{f}_i - \hat{\mathbf{f}}_i)^2\right) + \beta\left(\int_\Omega (\mathbf{u}-\hat{\mathbf{u}})^2 d\Omega\right). \quad (6.4)$$

This strategy facilitates a smooth transition between the two optimization steps by adjusting the relative importance of the pretraining and main training loss functions. In particular, the loss function $\mathcal{M}$ (Eq. 6.3) associated with the supervised pre-training continues to enforce a data-driven alignment and guarantees "continuity" in the optimization process. The term $\mathcal{J}$ (Eq. 6.1) corresponding to the loss function of the physics-constrained loop is introduced to minimize the mean flow reconstruction error.

The next section is dedicated to the discussion of the results. We show that the converged GNN model effectively predicts a forcing term $\mathbf{f}$ that is aligned with the ground truth and consistent with the physics of the system through the constraint introduced using the adjoint equations. At the same time, an effective model reconstructing the mean flow $\overline{\mathbf{u}}$ is learned. The method outperform the accuracy of standard techniques of mean flow reconstruction.

6.3 . Results

In this section, we present the improvements obtained using the proposed data assimilation scheme for the reconstruction of the mean flow field $\overline{\mathbf{u}}$. Tests are carried out by considering several scenarios, and in particular the reconstruction of the mean flow starting from noisy probes, incomplete flow fields (inpainting) and sparse measurements. The models are compared with the supervised learning method introduced in chapter 5 introduced as baseline reference, where the GNN model is trained by solely learning the forcing stress $\mathbf{f}$ based on the DNS data. This forcing stress $\mathbf{f}$ is then used as input to the RANS equations (Eq. 2.8b) in order to reconstruct the mean flow $\overline{\mathbf{u}}$. The GNN's objective is to minimize the discrepancy between the predicted and the ground truth forcing stress (Eq. 3.30), without any constraint introduced

based on the physics of the system. In contrast, the hybrid data assimilation scheme discussed in this chapter introduces in the training process of the GNN the physics constraint. To compare the two methods, we evaluate their training curves after the pre-training phase by identifying the minimum loss values reached by each model in the training process. The percentage improvement is then computed as follows:

$$\mathcal{I}(\%) = \frac{min(\mathcal{J}_{\text{Supervised}}) - min(\mathcal{J}_{\text{Physics constrained}})}{min(\mathcal{J}_{\text{Supervised}})} \cdot 10^2, \tag{6.5}$$

where $min(\mathcal{J}_{\text{Supervised}})$ and $min(\mathcal{J}_{\text{Physics constrained}})$ represent the minimum values of the loss function on the mean flow reconstruction (Eq. 6.1) for the baseline (pure supervised learning) and the adjoint based methods, respectively. In the following, we introduce different learning task by focusing on the technical features of the method and discussing the achieved improvements in terms of mean flow reconstruction.

6.3.1 . Proof of Concept

The first test case we consider is the flow field reconstruction when the input of GNN is the complete mean flow $\overline{\mathbf{u}}$ (and Reynolds number Re) defined on the entire computational domain Ω. This test case is introduced as proof of concept of the method. We consider two cases of increasing complexity. The first case is a flow developing past a 2D cylinder at Reynolds number of $Re = 150$. This case is well documented in literature and its time-averaged mean flow is shown in Fig. 6.2a. The training dataset only contains as input the mean flow $\overline{\mathbf{u}}$ and its corresponding forcing term $\mathbf{f}$ as GNN target. The training curves in Fig. 6.2b show that starting from the pre-training phase, the implementation of the approach described in this paper leads to a substantial improvement in the mean flow reconstruction. Specifically, the improvement attains the value of $\mathcal{I} = 58.59\%$.

The second case consists of a two side-by-side cylinders configuration, also known in literature as the 'flip flop' case, at Reynolds number $Re = 90$. Its RANS resulting mean flow is shown in Fig. 6.3a. The training curves for this case in Fig. 6.3b demonstrate an even more pronounced improvement, with a reduction of $\mathcal{I} = 82.90\%$ in the loss curve. The results indicate not only the broad adaptability of the proposed approach but also how, in more complex models, the underlying physics and governing equations play a crucial role in further increasing the accuracy of the GNN model's prediction.

6.3.2 . Generalization

In this section, we test the generalization capabilities of the learned model. The training dataset consists of three cases of 2D cylinder at Reynolds numbers of $Re = [90, 110, 130]$. On the other hand, the validation dataset includes

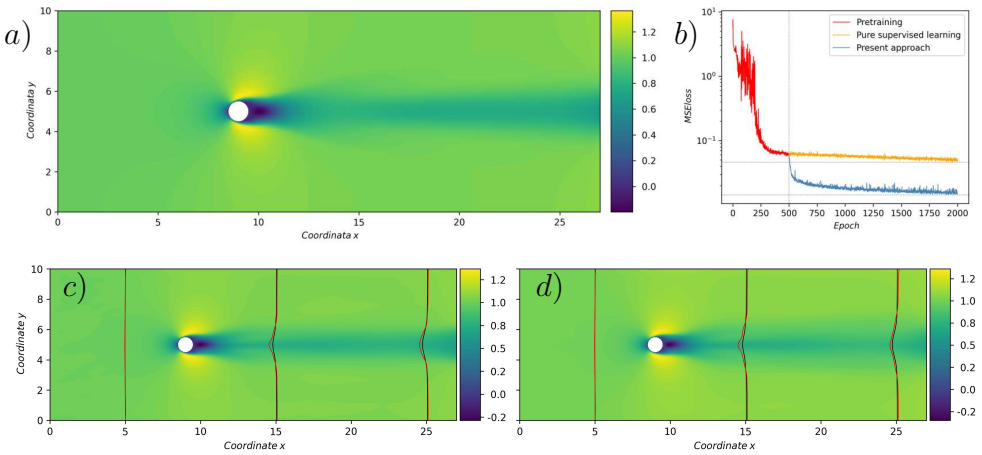


Figure 6.2: (a) The training mean flow input from the ground truth. The training dataset is composed by 1 meanflow-forcing pair at Reynolds number $Re = 150$; (b) the loss curves for the pure supervised approach (orange line) and the proposed approach (blue line) are shown. The two horizontal dotted lines indicate the minimum values of both curves, while the dotted vertical line indicates the end of the pre-training phase (Sec. 6.2.2); (c) the reconstructed mean flow from the pure supervised approach; (d) the reconstructed mean flow from the present approach. 1D line plots are overimposed on figures (c) and (d), comparing the predicted flow values (red line) with the ground truth (black line) at various sections along the flow field.

data points not included in the training set related to simulations of the flow around a 2D cylinder at Reynolds number $Re = 120$ as interpolation test, and $Re = 150$ aimed at testing the extrapolation properties. In Fig. 6.4a, the mean flow $\overline{\mathbf{u}}$ ground truth at $Re = 120$ case is shown. Based on the validation cases, we observe an improvement in the mean flow reconstruction by an average over the entire validation dataset of $\mathcal{I} = 73.27\%$. Specifically, we obtained an improvement of $\mathcal{I} = 78.96\%$ for the interpolation case at $Re = 120$, and $\mathcal{I} = 13.96\%$ for the extrapolation case at $Re = 150$. The improvement obtained on the training cases is $\mathcal{I} = 40.16\%$ as an average over the entire training dataset.

With this test case, the primary objective is to show that the presented approach enhances the generalization capabilities of the GNN model. To ensure clarity in our analysis, this generalization test case is deliberately isolated from the others. This separation allows maintaining a focused evaluation for each individual test case, targeting the specific goals of those tests without introducing confounding variables related to generalization.

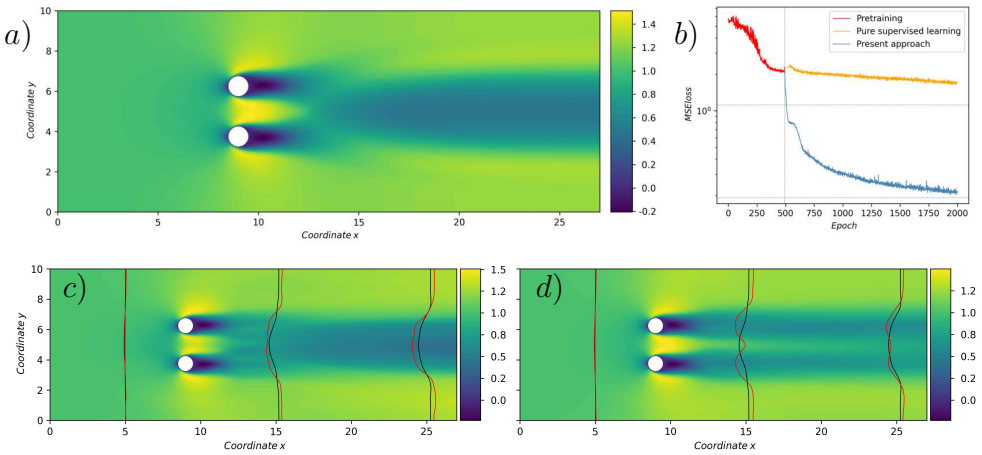


Figure 6.3: (a) The training mean flow input from the ground truth. The training dataset is composed by 1 meanflow-forcing pair at Reynolds number $Re = 90$; (b) the loss curves for the pure supervised approach (orange line) and the proposed approach (blue line) are shown. The two horizontal dotted lines indicate the minimum values of both curves, while the dotted vertical line indicates the end of the pre-training phase (Sec. 6.2.2); (c) the reconstructed mean flow from the pure supervised approach; (d) the reconstructed mean flow from the present approach. 1D line plots are overimposed on figures (c) and (d), comparing the predicted flow values (red line) with the ground truth (black line) at various sections along the flow field.

6.3.3 . Sparse Measurement

The learning task presented here involves the reconstruction of the mean flow on the entire computational domain using as input for the GNN measurements from randomly distributed probes. The training dataset is composed by two simulations of the flow past a cylinder for each Reynolds number in the range $Re = [90, 110, 130]$, resulting in six cases. For each case, 450 probes are placed in the mean flow stream, uniformly distributed across the entire computational domain Ω. Subsequently, 200 of these probes are randomly removed, leaving a sparse set of 250 probes. This sparse set of measurement on the mean flow $\overline{\mathbf{u}}$ is used as input to the GNN while its output prediction is compared with the corresponding forcing stress tensor from the DNS ground truth. Fig. 6.5a shows the random probes positioning on the mean flow, while Fig. 6.5b the average training curves on the training dataset. In this case, we demonstrate an improvement in the mean flow reconstruction across all the training cases by an average of $\mathcal{I} = 55.09\%$. This result highlights the robustness of the proposed approach in scenarios with sparse and randomly distributed measurements.

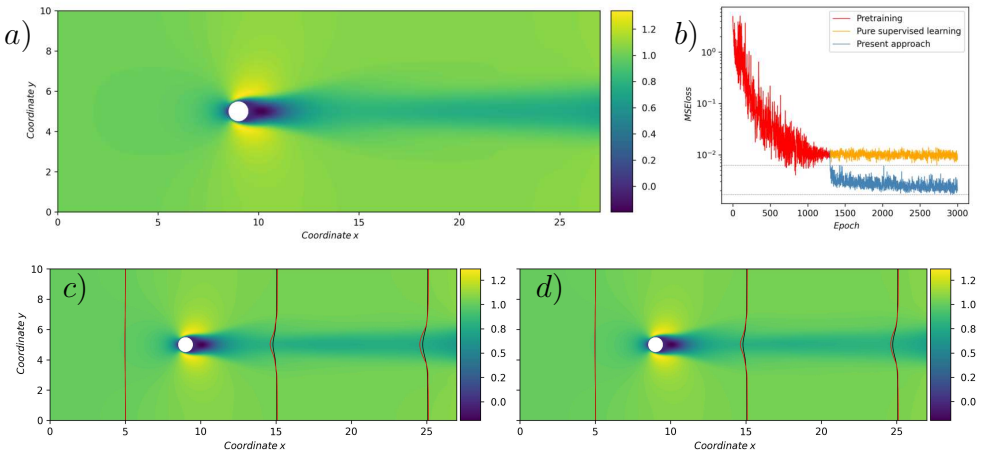


Figure 6.4: (a) The training mean flow input (at $Re = 120$) from the ground truth. The training dataset is composed by 3 meanflow-forcing pair at Reynolds number $Re = [90, 110, 130]$ while the validation dataset contains cylinder cases at $Re = [120, 150]$; (b) the loss curves for the pure supervised approach (orange line) and the proposed approach (blue line) are shown. The two horizontal dotted lines indicate the minimum values of both curves, while the dotted vertical line indicates the end of the pre-training phase (Sec. 6.2.2); (c) the reconstructed mean flow (at $Re = 120$) from the pure supervised approach; (d) the reconstructed mean flow (at $Re = 120$) from the present approach. 1D line plots are overimposed on figures (c) and (d), comparing the predicted flow values (red line) with the ground truth (black line) at various sections along the flow field.

6.3.4 . Denoising

In this test case, the input mean flow field is perturbed with a Gaussian noise. The probability density function used for the Gaussian distribution used to generate the noise is given by

$$\psi(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(z-\mu)^2}{2\sigma^2}}, \tag{6.6}$$

where z is the random variable, μ is the mean value of the normal distribution and σ represents its standard deviation. In this case we assumed $\mu = 0$, namely a standard normal distribution. The training dataset consists of three cases of cylinder flows, at Reynolds number $Re = [90, 110, 130]$, perturbed with Gaussian noise having $\sigma = [0.6, 0.4, 0.2]$, respectively. Fig. 6.6a shows the effect of $\sigma = 0.4$ Gaussian noise on the mean flow (at $Re = 110$) while Fig. 6.6b presents the accuracy in the mean flow reconstruction. The goal here is to remove the Gaussian noise and accurately reconstruct the denoised mean flow field. Our approach demonstrates an improvement on the training dataset

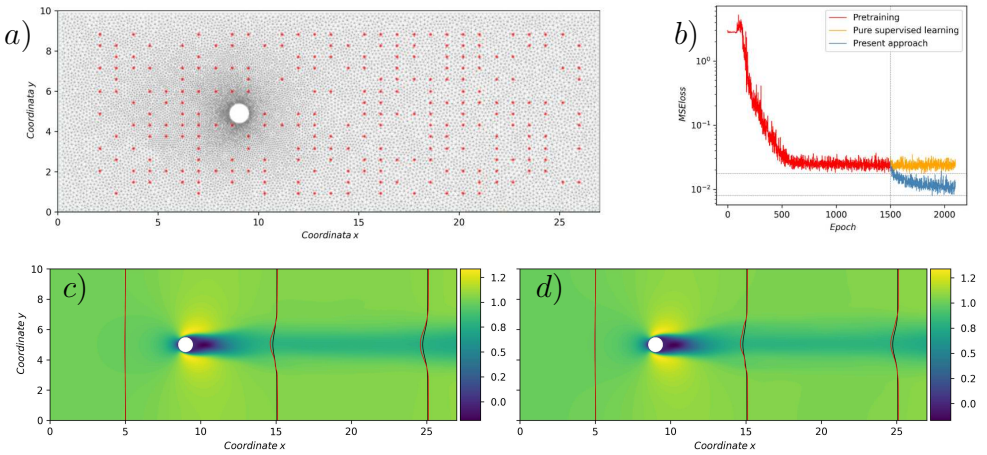


Figure 6.5: (a) An example of the probes positioning on the mean flow. The training dataset is composed by 6 mean flow-forcing pairs at Reynolds number in the range $Re = [90, 110, 130]$ (two instances for each case) with 250 randomly distributed probes; (b) the loss curves for the pure supervised approach (orange line) and the proposed approach (blue line) are shown. The two horizontal dotted lines indicate the minimum values of both curves, while the dotted vertical line indicates the end of the pre-training phase (Sec. 6.2.2); (c) the reconstructed mean flow (at $Re = 110$) from the pure supervised approach; (d) the reconstructed mean flow (at $Re = 110$) from the present approach. 1D line plots are overimposed on figures (c) and (d), comparing the predicted flow values (red line) with the ground truth (black line) at various sections along the flow field.

by a factor of $\mathcal{I} = 45.67\%$ as an average over the training cases.

6.3.5 . Inpainting

In this test, masking patches are randomly applied to the input mean flow field. The training dataset consists of three cases of cylinder obstacle at Reynolds number $Re = [90, 110, 130]$, each with different patch locations (Fig. 6.7a). The goal is to reconstruct the mean flow field by filling in the missing patches. The approach demonstrates improvements on the training cases by an average of $\mathcal{I} = 41.73\%$, successfully restoring the missing portions of the field and enhancing the overall reconstruction accuracy.

6.3.6 . Discussion and outlooks

In this section, we introduced a hybrid data-assimilation for the reconstruction of the mean flow, starting from corrupted or incomplete data. By integrating RANS equations into the GNN training process through an adjoint optimization framework (Sec.4), our model demonstrates superior accuracy

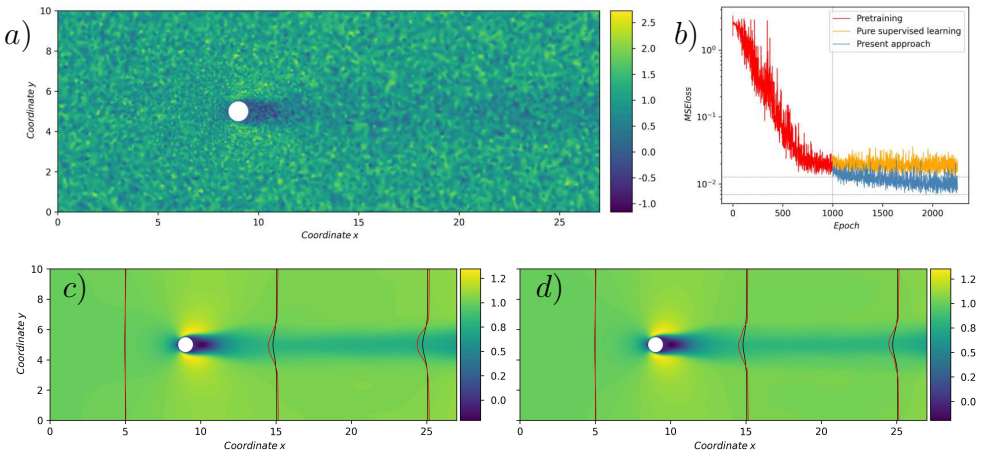


Figure 6.6: (a) Gaussian perturbed mean flow (at $Re = 110$). The training dataset is composed by 3 mean flow-forcing pairs at Reynolds number $Re = [90, 110, 130]$ perturbed with a Gaussian noise having $\mu = 0$ and $\sigma = [0.6, 0.4, 0.2]$, respectively; (b) the loss curves for the pure supervised approach (orange line) and the proposed approach (blue line) are shown. The two horizontal dotted lines indicate the minimum values of both curves, while the dotted vertical line indicates the end of the pre-training phase (Sec. 6.2.2); (c) the reconstructed mean flow (at $Re = 110$) from the pure supervised approach; (d) the reconstructed mean flow (at $Re = 110$) from the present approach. 1D line plots are overimposed on figures (c) and (d), comparing the predicted flow values (red line) with the ground truth (black line) at various sections along the flow field.

in reconstructing mean flows, outperforming purely data-driven models. The proposed method takes mean flow inputs under varying conditions, such as noisy, sparse measurements or patch—masked flows, and predicts the closure term of the RANS equations. This predicted term is then used to solve the RANS equations and reconstruct a complete, uncorrupted mean flow. The use of adjoint methods for computing the gradients of the loss function allows the GNN to incorporate physical knowledge into its training process and enhances results' accuracy when compared to the supervised learning strategy introduced in chapter 6.

The study offers numerous possibilities for future research. First of all, the introduction of a numerical solver represents also a bottleneck, as the solution of the direct and adjoint RANS equations is required. The performance of the entire method highly depends on the available computational resources and the efficiency of the numerical solver used. Improvements can be achieved by efficient, parallel FEM code. This would enable to test the ap-

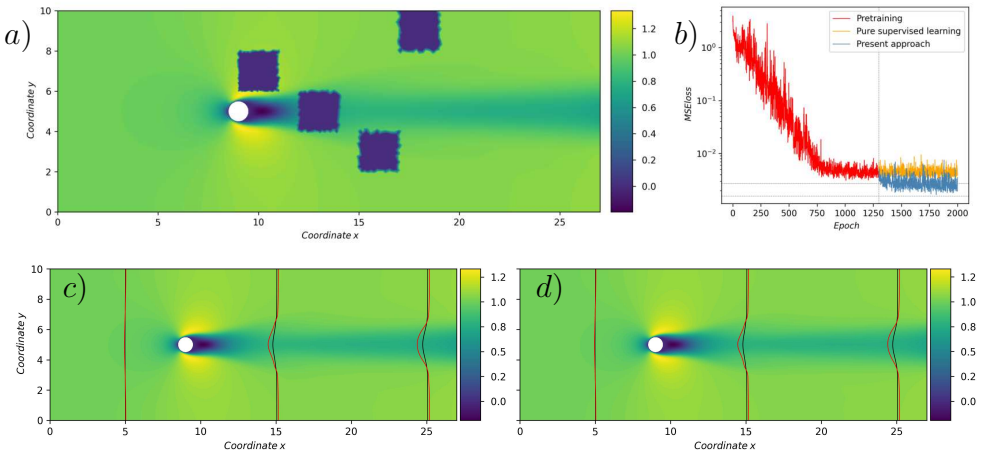


Figure 6.7: (a) Patch mask applied on the mean flow (at $Re = 110$). The training dataset is composed of 3 mean flow-forcing pairs at Reynolds number $Re = [90, 110, 130]$ with randomly located patching mask; (b) the loss curves for the pure supervised approach (orange line) and the proposed approach (blue line) are shown. The two horizontal dotted lines indicate the minimum values of both curves, while the dotted vertical line indicates the end of the pre-training phase (Sec. 6.2.2); (c) the reconstructed mean flow (at $Re = 110$) from the pure supervised approach; (d) the reconstructed mean flow (at $Re = 110$) from the present approach. 1D line plots are overimposed on figures (c) and (d), comparing the predicted flow values (red line) with the ground truth (black line) at various sections along the flow field.

plication of the current data assimilation scheme to more complex 3D cases, including turbulent flows at higher Reynolds numbers. Test cases of higher complexity would provide valuable insights into the applicability to realistic cases at larger scales.

From the ML viewpoint, a multi-scale prediction process can be envisioned where a series of GNNs is introduced at different resolutions aimed at refining progressively the closure term predictions. For instance, one may introduce an initial GNN model predicting the forcing stresses on a coarse or sparse grid, followed by models refining the prediction at finer scales, as done with super-resolution techniques.
Moreover, additional physics-informed elements could be added into the loss function. Beyond the RANS equations, the model could include explicit terms associated with boundary conditions, such as the inflow or outflow profiles, ensuring that the predicted flows better represent physical expectations.

Finally, one could consider alternative ML models to the GNNs such as the transformers [Vaswani et al., 2017]. In contrast with GNN models, transformers are highly effective in capturing dependencies between widely separated

nodes. This attribute is particularly valuable when dealing with sparse or irregularly distributed measurements, as it allows the model to identify spatial dependencies in the flow field.

7 - Part III: Shape optimization of Ducted Wind Turbines (DAWT)

7.1 . Introduction

In the field of CFD, traditional optimization methods (Sec. 4.4) face significant challenges, particularly when applied to complex, nonlinear and multi-parametric systems. These methods often depend on iterative evaluations of the objective function, which, when this latter depends on numerical simulations, become computationally expensive [Jameson, 1995, Rao, 2009]. Moreover, these methods struggle to perform effectively in scenarios with large and complex parameter spaces, where the relationship between design variables and performance metrics is often highly non-linear. In other words, the high computational cost of each numerical simulation, combined with the need for numerous iterations, makes traditional approaches infeasible for optimization tasks that demand the exploration of a wide design space. These limitations hinder the ability to achieve optimal solutions within a reasonable time frame. An example is given by the optimization of energy production flow systems, in which the maximization of the desired output (i.e., the power production, or the overall efficiency of the system, among others) depends on the complex dynamics of an infinite-dimension system characterized by an often chaotic and three-dimensional behaviour such as the fluid flow which invests the machine. A notable example are wind turbines, whose performance optimization is based on the non-trivial dynamics of the flow which invests them, which is inherently three-dimensional and turbulent (Porté-Agel et al. [2020], De Cillis et al. [2022]), as well as on the numerous parameters describing the geometrical configuration of each element of the system (namely, the blade airfoils and their radial development, the tower and nacelle, etc..).

In this framework we tackle, using ML techniques, the geometry optimization of an element of a newly developed wind energy system for urban use, the Diffuser-Augmented Wind Turbine (DAWT).

DAWTs (Fig. 7.2) are an advanced class of wind turbines that differ from traditional Open-Rotor Wind Turbines (OWT) by incorporating a surrounding diffuser or duct, designed to enhance the flow of wind through the rotor. This configuration results in an increase in power output due to what is known as the *diffuser effect*, exceeding the Lanchester-Betz limit for conventional Horizontal Axis Wind Turbines (HAWT) [Bontempo and Di Marzo, 2023]. The Lanchester-Betz limit, formulated by German physicist Albert Betz in 1919, states that no wind turbine can convert more than 59.3% of the kinetic energy of the wind into mechanical energy.

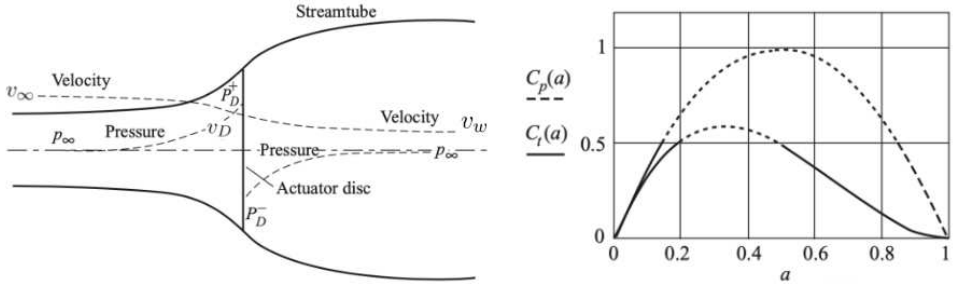


Figure 7.1: (Left) Representation of the *actuator disk* configuration for a wind turbine. (Right) Power coefficient C_p as a function of the axial flow induction factor a for a wind turbine.

This theoretical maximum arises in the context the *actuator disk* model, which is a theoretical approach used to represent the time-averaged behavior of a wind turbine. It conceptualizes the turbine as a uniformly permeable disk that extracts energy from the wind flow, without explicitly modeling the blades.
The actuator disk model assumes that the wind slows down as it approaches the rotor plane, where energy is extracted. With reference to Fig. 7.1(Left), denoting the undisturbed velocity of the wind far upstream as v_∞, the velocity at the rotor plane v_D can be expressed in terms of the axial flow induction factor a as:

$$v_D = v_\infty(1 - a) \tag{7.1}$$

The energy extraction also results in a reduction of wind velocity downstream v_w of the turbine. The velocity far downstream, v_w, can be derived as:

$$v_w = (1 - 2a)v_\infty. \tag{7.2}$$

Given the velocity difference between up and down far stream, an axial thrust T exerted on the air by the rotor can be defined as:

$$T = (v_\infty - v_w)\rho A v_D \tag{7.3}$$

This force on the air flow comes from the rotor, or actuator disk, and it's due to the pressure difference across the disk. Therefore, it can be seen as:

$$(p_D^+ - p_D^-)A = (v_\infty - v_w)\rho A v_\infty(1 - a). \tag{7.4}$$

where v_D is expressed as reported in Eq. 7.1. Combining Eq. 7.4 and Eq. 7.2 and multiplying by v_D, the power extracted from the wind can be written as:

$$P = T v_D = 2\rho A v_\infty^3 a(1 - a)^2 \tag{7.5}$$

To quantify the efficiency of the energy extraction, we define the power coefficient C_p (Fig. 7.1(Right)):

$$C_p = \frac{P}{\frac{1}{2}\rho A v_\infty^3} \tag{7.6}$$

where the denominator $\frac{1}{2}\rho A v_\infty^3$ represents the total available power in the wind. Plugging Eq. 7.5 in Eq. 7.6 gives:

$$C_p = 4a(1-a)^2 \tag{7.7}$$

To determine the maximum C_p, we take the derivative of C_p with respect to the axial flow induction factor a and set it equal to zero:

$$\frac{dC_p}{da} = 4(1-a)(1-3a) = 0 \tag{7.8}$$

Solving this equation gives:

$$a = \frac{1}{3} \tag{7.9}$$

Substituting $a = \frac{1}{3}$ back into the expression for C_p, we get:

$$C_p = \frac{16}{27} \approx 0.593 \tag{7.10}$$

This theoretical maximum efficiency arises because the wind must retain enough energy to keep flowing past the turbine, ensuring a continuous flow.

In order to overcome this theoretical limit, research have recently developed the DAWTs concept, where the diffuser accelerates the airflow through the rotor and creates a region of lower pressure downstream that draws additional air through the rotor. This effect allows DAWTs to produce more power than conventional open-rotor turbines of comparable rotor size [van Bussel, 2007, Ilhan et al., 2021]. Other advantages of DAWTs are the lower cut-in speed, sensitivity to yaw angles, noise level, tip speed losses, and a higher safety footprint in case of mechanical failure of the rotor. All these aspects make DAWTs particularly suitable for small-size applications in urban environments [Dilimulati et al., 2018, Stathopoulos et al., 2018, Hassanli et al., 2019, Potsis et al., 2023]

Small changes in the duct geometry, particularly in the diffuser angle and its placement relative to the rotor, can lead to significant variations in DAWT performance, making it an ideal candidate for shape optimization. Indeed, optimization of DAWTs has been extensively explored in the literature, focusing particularly on how geometric modifications to the diffuser or duct can impact the turbine performance. Foreman et al. [1978] explored these effects by conducting experiments on diffuser-augmented models, highlighting how specific duct orientations and geometries could improve wind turbine efficiency.

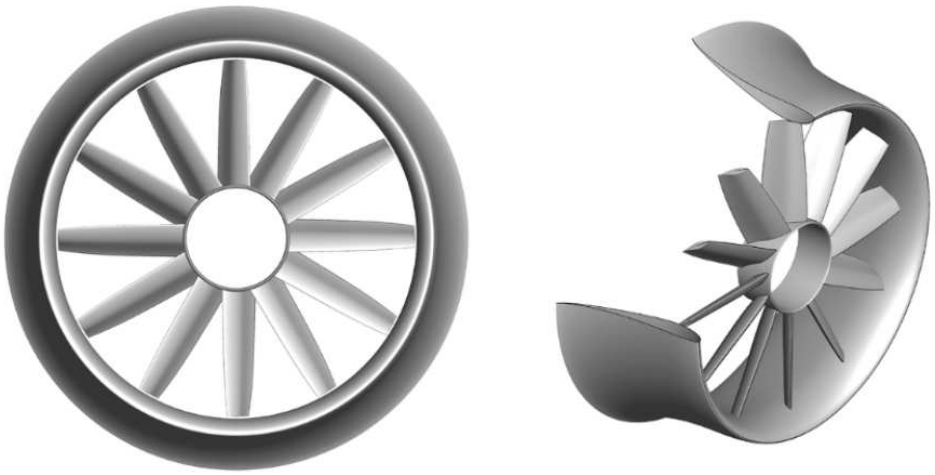


Figure 7.2: (Left) Frontal view of a Diffuser-Augmented Wind Turbine (DAWT) and its perspective view (Right). Figure inherited from Bontempo and Manna [2022]

Aranake et al. [2015] advanced the field by applying three-dimensional CFD analyses to various shrouded turbine configurations, focusing on the effects of boundary layer separation and vortex shedding within the duct. Their work showed that flow separation along the inner surfaces of the duct could reduce efficiency, indicating the need for precise duct shape optimization to mitigate these effects. Venters et al. [2016] introduced optimization techniques based on RANS simulations to identify optimal duct configurations for enhanced energy capture. Their approach used high-fidelity CFD models to iteratively refine duct shapes, emphasizing the computational challenges of such optimization processes due to the large number of design variables and the need for extensive computational resources.
These studies collectively underscore the significant computational cost associated with duct shape optimization for DAWTs as many of these approaches rely on iterative CFD simulations. Each shape modification or adjustment in duct orientation requires, indeed, a completely new full simulation: an extensive analysis of all possible geometric configurations would result in a high-dimensional, nonlinear design space, requiring a large number of simulations, and thus leading to impractical computational costs.

To overcome these limitations, in this work we introduce a GNN as a surrogate model, in order to quickly evaluate the flow fields associated with different duct geometries. This surrogate model, trained on high-fidelity CFD data (Sec. 7.2), enables a quick evaluation of the performance metrics, avoiding the need of RANS simulations at each iteration. The chosen metric is the power coefficient (C_p). In principle, simpler NN architectures can be used for

the prediction of a scalar output, without necessary including geometric information. In practice, although C_p remains a crucial performance metric, it alone cannot capture the interactions in the flow field, which are essential for assessing variables that impact downstream conditions. For instance, the prediction of the full flow field enables for broader optimization targets, such as the reduction of the turbulence intensity and the enhancement of the flow regularity downstream of the turbine, that are critical in wind farms, where the wake from an upstream turbine can substantially affect the performance of downstream turbines, or else, in urban applications, where they impact on the local population or ecosystem, due to noise or unsteady airflow. Moreover, predicting the whole flow field allows to avoid critical flow configurations indicating an incipient flow separation on the diffuser, which may result in a performance drop for a minimal change of the flow conditions. In this sense, an architecture predicting the entire flow field offers significant advantages in addressing these optimization objectives. Finally, it is worth stressing that the choice of GNN meets the need of a detailed, geometrical parametrization of key turbine components, such as the duct shape and diffuser position.

In this work, the optimization process is carried out by pairing the surrogate model based on GNN with a Reinforcement Learning (RL) algorithm [Sutton and Barto, 2018], which acts as an agent optimizing the duct's geometry by adjusting control parameters iteratively based on the feedback from the surrogate model used as the environment. RL is an umbrella term covering numerous strategies and algorithms aimed at maximizing or minimizing a reward or cost-function by learning policies from interactions with an environment that is not necessarily known a priori. Recent results are showing that these techniques are particularly suitable for problems involving non-linear environments, including active control of fluid flows and problems where standard techniques of optimization can be limited due to uncertainties in the modeling step [Rabault et al., 2019, Bucci et al., 2019].

In this part of the thesis, we consider a proof of concept for demonstrating the feasibility of the approach by focusing on a single geometric parameter: the angle of the duct α. The RL algorithm is used to optimize this parameter to maximize the power coefficient (C_p). Although this is a simplified case, the goal is to showcase the potential of combining a GNN-based surrogate model replacing a full-scale numerical simulation with RL algorithms for the optimization.

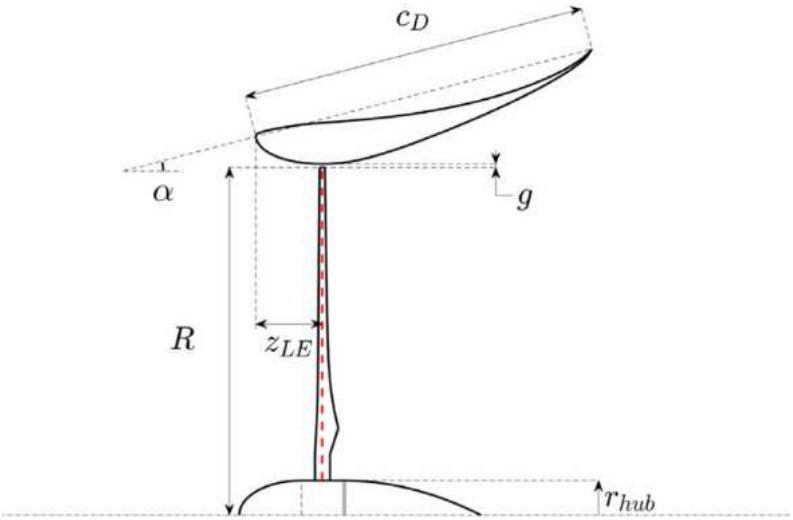


Figure 7.3: Sketch of a ducted wind turbine DAWT reporting the relevant geometrical parameters. Figure adapted from Bontempo and Di Marzo [2023].

7.2 . DAWT Ground Truth Data and Numerical Setup

The CFD flow fields used in this work for the optimization of DAWTs duct's shape are a series of RANS simulations provided by R. Bontempo [1]. These RANS simulations provide a rich and comprehensive dataset for DAWT performance analysis, with particular focus on the influence of duct geometry on the wake structure and turbine efficiency. These simulations data serve as training dataset for the GNN employed in the optimization stages.

In the following, we quickly summarize the main features of the flow under consideration, outlined in the work by Bontempo and Di Marzo [2023]. Fig. 7.3 shows a sketch of a DAWT, including the main geometrical parameters describing its geometry. Specifically, α represents the stagger angle of the duct (the control parameter in this work), C_D is the chord of the duct airfoil shape, g is the rotor-duct tip gap, R the rotor radius, r_{hub} the hub radius and z_{LE} the leading edge (LE) axial coordinate. The aerodynamic flow of the DAWT is simulated using a coupled Blade Element Theory (BET) approach [Burton et al., 2011]. In this method, Blade Element Theory (BET) is employed to model the rotor's aerodynamics, while simultaneously incorporating the effects of the duct. This coupled approach ensures that the aerodynamic interactions between the rotor and the duct are fully captured, leading to a more accurate and realistic representation of the complex flow dynamics in the CFD simulation. The computational domain is a cylindrical volume designed to

[1] Prof. at Dipartimento di Ingegneria Industriale, Università degli Studi di Napoli Federico II, Naples, Italy

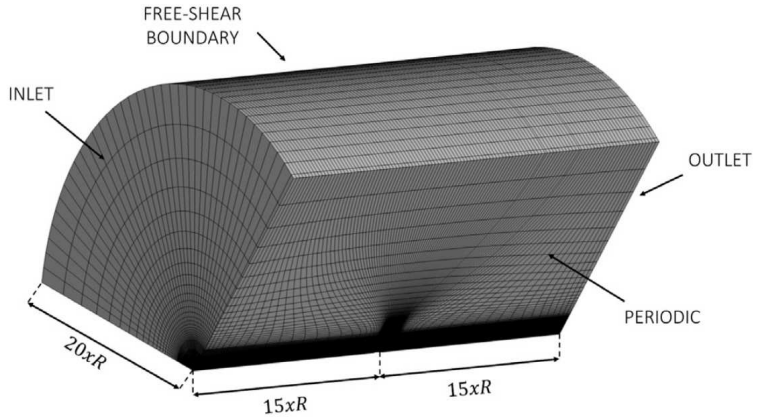


Figure 7.4: Representation of a portion of the computational domain along with the adopted boundary conditions. Figure adapted from Bontempo and Di Marzo [2023]

capture the flow field around the DAWT while ensuring minimal boundary interference, so that the computational results accurately reflect the aerodynamic interactions without significant distortions from artificial boundary effects. To this end, the domain extends both upstream and downstream, with the inlet and outlet boundary set 15 rotor radii R upstream and downstream from the rotor plane, respectively. The radial boundary extends to 20 rotor radii R, balancing computational feasibility with the need for accurate far-field boundary conditions (Fig. 7.4). The boundary conditions are tailored to reflect realistic operating conditions for DAWTs. At the inlet boundary, a uniform velocity profile, V_∞, simulates the ambient wind. At the outlet boundary, a zero static pressure condition with radial equilibrium is imposed. The outer radius boundary is treated as a free-shear wall, while periodic conditions are applied to the lateral surfaces.

The mesh is divided into distinct regions for the rotor, duct, and external flow to optimize grid density where it is most critical. The mesh employs structured grids for the rotor and duct regions, with dense clustering near blade surfaces and the duct's inner walls. This localized refinement is crucial for resolving boundary layer dynamics, flow separation, and other near-surface phenomena that influence the ducted turbine's performance. The rotor mesh, developed by stacking 2D grids in the spanwise direction, allows for detailed representation of blade curvature and twist. Mesh independence is validated by monitoring the convergence of C_p and radial force distributions across multiple grid densities (Fig. 7.5), with grid refinement concentrated in the wake region; only minimal changes were observed between the medium and fine grids, confirming the mesh's adequacy for performance analysis.

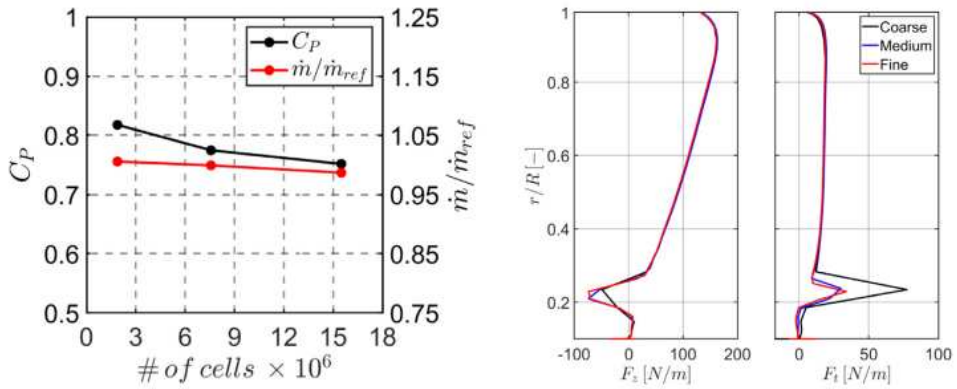


Figure 7.5: Grid independence test. (Left) Power Coefficient C_p and mass flow rate ingested by the rotor (Right) Linear density of the axial and tangential forces. Figure taken from Bontempo and Di Marzo [2023]

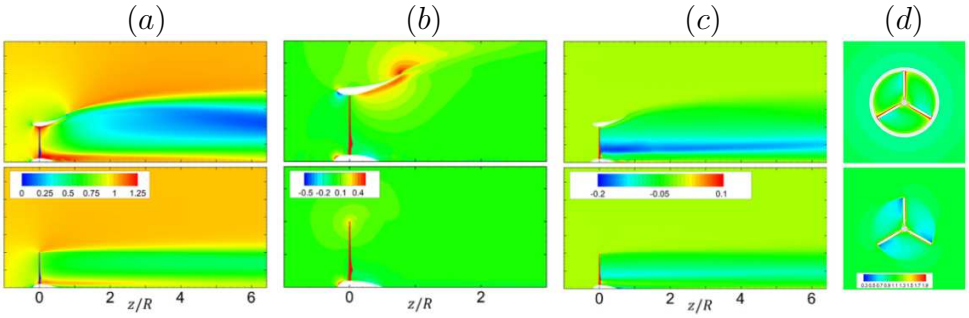


Figure 7.6: Comparison between the ducted (DAWT) (Top) and open (OWT) (Bottom) configurations for $V_\infty = 7^{m}/_{s}$: (a) azimuthally-averaged normalized axial velocity, (b) azimuthally-averaged normalized radial velocity contours, (c) azimuthally-averaged normalized tangential velocity contours, (d) normalized axial velocity contours at the rotor plane. Figure adapted from Bontempo and Di Marzo [2023]

RANS equations are numerically solved using Ansys Fluent's [ANSYS Inc., 2023] Finite Volume approach. Convective fluxes are discretized with a second-order upwind scheme, and diffusive terms use a central-difference scheme to maintain accuracy. To model turbulence, the κ-ω Shear Stress Transport (SST) model by Menter et al. [2003] is selected for its suitability in wind turbine applications, given its balance of accuracy in boundary layer regions and computational efficiency. This model is particularly effective in resolving the separation and reattachment behaviors along the duct surfaces and the rotor blades, which directly impact DAWT performances.

The resulting flow field is shown in Fig. 7.6 for a freestream velocity $V_\infty = 7^{m}/_{s}$; this velocity is adopted for all the simulations used for creating the datasets, ensuring uniformity while varying other parameters, such as the angle α of the

duct. These flow fields are used as training dataset for the GNN in the next optimization stages, as detailed in Sec. 7.3.

7.3 . Optimization Loop

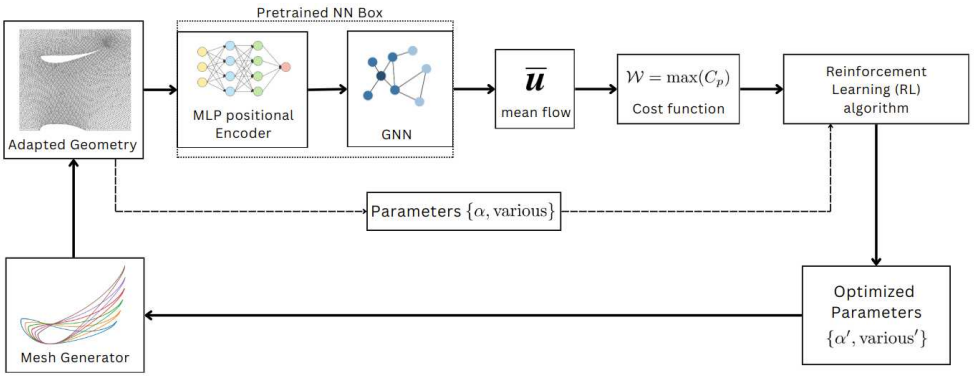


Figure 7.7: End-to-end optimization loop; the complete mesh along with the control variable α are the NN's block input (Sec. 7.4); $\overline{\mathbf{u}}$ is the GNN's predicted mean flow; C_p is power coefficient to maximize.

The optimization process is illustrated in the diagram in Fig. 7.7, which highlights the interactions between each step in the loop. The aim of this iterative cycle is to refine the duct's angle α in order to maximize the power coefficient (C_p). Below, a brief overview of the loop is provided.

- Mesh input: The loop begins by feeding a mesh of the DAWT, with an initial duct angle α, into a pre-trained NN block (Sec. 7.4), which has been trained to predict the flow fields, given a certain mesh.
- Flow Field Prediction: The pre-trained NN model (Sec. 7.4) takes this mesh as input, and provides as output the predicted flow fields, thus replacing in this step a full-order numerical simulation. The NN acts as surrogate model of the system.
- C_p computation: The predicted flow fields are used to calculate the power coefficient (C_p) (Sec.7.5), which is the performance metric we aim to maximize.
- Optimization – Reinforcement Learning (RL): The calculated C_p, along with the geometric parameters of the duct (in this case, the angle of the duct α), are provided to a RL algorithm (Sec. 7.6). The RL agent evaluates these inputs and selects improved geometric parameters with the goal of increasing C_p.

- Mesh Generation: The optimized parameters chosen by the RL agent are then passed to a mesh generator, which creates a new mesh based on the updated geometry.

The new mesh is fed back into the NN surrogate model (Sec. 7.4), and the loop continues until an optimal C_p is achieved. This optimization cycle iterates until the system converges on an optimal duct geometry that maximizes the C_p. One of the significant advantages of using a GNN-based surrogate model in combination with RL is the reduction in computational cost. The ground truth, which relies on RANS simulations, requires approximately 20 minutes per simulation, executed on 5 cores of an Intel(R) Core(TM) i9-10900X CPU @ 3.70GHz. In contrast, the GNN surrogate model can provide an approximation of the flow field in just a few seconds on a single core of an Intel(R) Core(TM) i5-1135G7 @ 2.40GHz. This strong reduction in computational time allows for more efficient exploration of the design space and makes RL-based optimization practical for realistic applications or when computational resources are limited.

7.4 . Machine Learning (ML) in the Optimization Loop

This chapter delves into the details of the NN pre-trained block within the optimization loop (Fig. 7.7). The NN block consists of two primary components: a MLP Positional Encoder and a GNN. These two NNs operate in tandem to predict the flow fields for different wind turbine duct configurations, providing a fast and efficient surrogate model that can replace computationally intensive CFD simulations. Notably, the GNN used in this work shares the same hyperparameters as the one used in the previous chapters, detailed in Sec. 3.8.3.

The optimization process starts when the mesh of the wind turbine is fed into the NN block. However, one of the key challenges in this setup is ensuring that the GNN can generalize to unseen configurations, as different geometries and mesh topologies may significantly affect the predictive accuracy of the model. Directly inputting the raw coordinates of the mesh points into the GNN would result in a GNN model that is highly sensitive to the specific arrangement of the nodes. The model tends to learn a specific mapping between these fixed coordinates in the training dataset and the associated flow field, becoming overly dependent on the precise arrangement of the input nodes and therefore loosing its inherent invariance to nodes permutation. In other words, the GNN becomes "fixed" to particular spatial arrangements from the training dataset, and therefore it struggles to generalize when tested on new mesh arrangements. This sensitivity can lead to poor generalization, especially when the mesh structure changes.
To overcome this limitation, we use an MLP Positional Encoder as a pre-processing

step. The MLP transforms the raw coordinates of the mesh points into a latent, non-physical representation. This transformation captures the important positional information of the mesh points while abstracting away the exact permutation of these points. The MLP effectively creates a latent space representation that decouples the GNN from the specific arrangement of nodes, enabling the GNN to operate independently of the mesh structure. Once the mesh coordinates have been encoded by the MLP, the resulting latent representations are fed into the GNN. The GNN is then responsible for predicting the flow fields based on the latent representation of the mesh.

This combination of the MLP and GNN allows the network block to handle a wide range of mesh configurations and generalize effectively to new geometries. The MLP ensures that the input data is processed in a consistent way, while the GNN exploits the encoded positional relationships to predict the flow fields.

7.4.1 . NNs pre-training

A critical aspect of the whole optimization loop is the pre-training of both the Positional MLP and the GNN. Before these NNs can be used in the optimization process, they must be trained on ground truth data obtained from CFD simulations (Sec. 7.2), learning to infer the flow fields given a mesh as input. The pre-training is conducted in a supervised learning environment on data obtained from RANS simulations. From these simulations, a subdomain of size $1 \times R$ upstream, $2 \times R$ downstream, and $2 \times R$ in the radial direction (where R is the rotor radius) is extracted. This subdomain is chosen to reduce the computational load of training the NNs, as the calculation of the power coefficient (C_p), which we aim to optimize, only requires the flow field in the immediate vicinity of the rotor. Nonetheless, future works aimed at optimizing the far field will require an extended domain for the training process.

The training dataset consists of flow fields for duct configurations with angles α ranging from $5°$ to $25°$ with $\Delta\alpha = 5°$, excluding $\alpha = 15°$, which was reserved for validation purposes. A sketch representation of the possible duct positioning in the training dataset is given in Fig. 7.10. These angles represent variations in the duct geometry, starting from a baseline configuration where $\alpha = 0$. The variations are generated by applying a rigid rotation to the duct using a rotation matrix centered at the throat point of the duct-rotor system. The resulting flow fields corresponding to these rotated geometries serve as the ground truth for the supervised learning process. During training, the two NNs, the Positional MLP and the GNN, are trained together to learn the mapping between the input mesh and duct angle to the corresponding flow fields. This pre-training phase is conducted independently of the optimization loop to ensure that the networks can make accurate predictions when included in the wider optimization framework. Fig. 7.8(a) shows the flow fields sub-

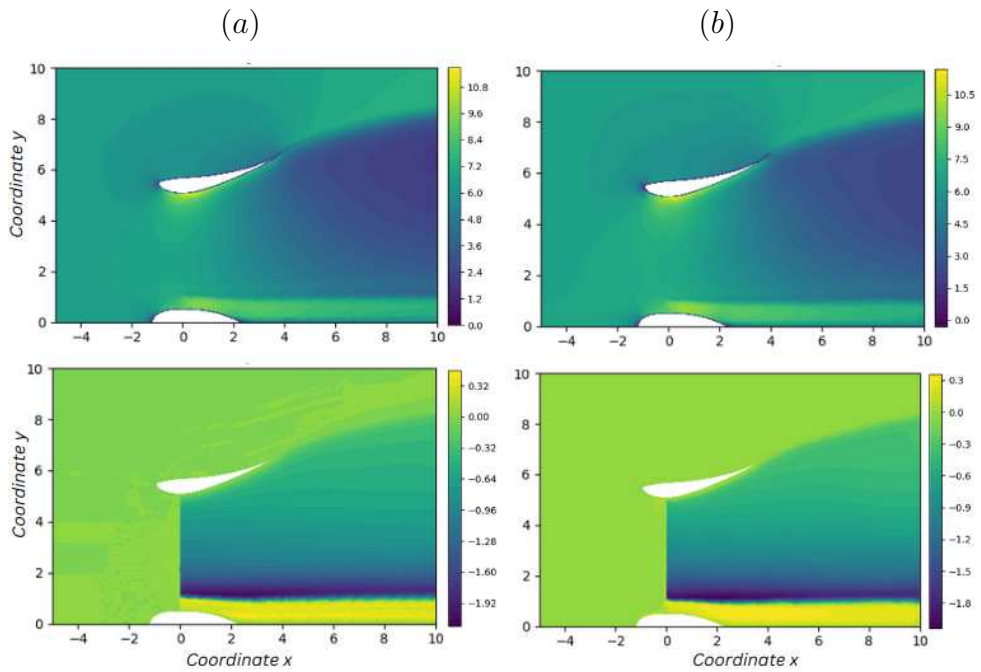


Figure 7.8: Comparison of flow fields for $\alpha = 15°$ (not included in the training dataset). (a) the ground truth flow fields, (b) the GNN flow fields prediction, (Top) axial velocity component, (Bottom) swirl velocity component.

domain extraction as it comes from the ground truth, while Fig. 7.8(b) their relative NNs prediction for an angle α not included in the training dataset, specifically $\alpha = 15°$. The accuracy of the reconstruction demonstrates the network's ability to generalize to unseen cases included in the interval of parameters.

7.5 . The Optimization Cost Function

The optimization loop aims to maximize the power coefficient (C_p) of the wind turbine. Specifically, the power coefficient is calculated using the Euler equations for turbomachinery, introduced in the following (Eq. 7.14). This approach provides an efficient method for approximating the power coefficient C_p based on the flow characteristics upstream and downstream of the *rotor disk*. As previously discussed, this model is primarily based on momentum theory, which considers the conservation of mass, momentum, and energy across the control volume. While this approach is relatively simplified compared to more detailed models like the Blade Element Theory (BET), it is computationally inexpensive and well-suited for rapid iterative optimization.
Within the *rotor disk* context, the power extracted by the wind turbine can be derived using the Euler's equations for turbomachinery, which describe

the relationship between the flow velocity components and the energy extracted. The basic principle relies on the conservation of angular momentum through an infinitesimal annular ring of area $2\pi r dr$, which induces a corresponding torque. Since both axial (i.e., v_a) and tangential (i.e., v_θ) components of the flow velocity depend on the considered radius, the mechanical power extracted by each infinitesimal annulus of the wind turbine rotor (dP) can be written as:

$$dP(r) = d\dot{m}(r)U(r)\Delta v_\theta(r) \tag{7.11}$$

where $d\dot{m}$ is the mass flow rate through the considered annulus; U is the linear velocity of the rotor's blade at the considered radius; Δv_θ is the difference in tangential (or swirl) velocity between the upstream and downstream flow across the actuator disk. The mass flow rate $\dot{m}$ is defined as

$$d\dot{m} = 2\pi r \rho v_a dr, \tag{7.12}$$

in which ρ is the air density, $2\pi r dr$ is the cross-sectional area of the annulus and v_a is the axial velocity of the flow through the considered annulus. Plugging Eq. 7.12 into Eq. 7.11, we get the equation that represents the mechanical power extracted from the wind by an annulus of the rotor, where the key parameters include the mass flow rate, axial velocity, tangential velocity difference, and the rotor's angular velocity

$$dP = 2\pi r \rho v_a U \Delta v_\theta dr. \tag{7.13}$$

To obtain the total power extracted by the rotor, we must integrate Eq. 7.13 along the entire blade length from the hub (r_{hub}) to the tip (r_{tip}). In fact, the contribution of each blade element to the total power is, indeed, not uniform, as the tangential and axial velocity change along the blade. The power extraction over the entire blade length is given by

$$P = \int_{r_{\text{hub}}}^{r_{\text{tip}}} \rho v_a(r) r \omega \Delta v_\theta(r) 2\pi r dr, \tag{7.14}$$

where ω is the angular velocity of the rotor and r is the local radius of the actuator disk. This integral equation (Eq. 7.14) accounts for the fact that the blade speed (related to $r\omega$), the tangential flow velocity difference and the axial velocity vary with the radial position.
Finally, the power coefficient (C_p) quantifies the efficiency of the turbine in converting wind energy into mechanical power. It is defined as the ratio of the power extracted by the turbine P (Eq. 7.14) to the total available power in the wind P_{wind}. The total available power in the wind is expressed as

$$P_{\text{wind}} = \frac{1}{2}\rho A v_\infty^3, \tag{7.15}$$

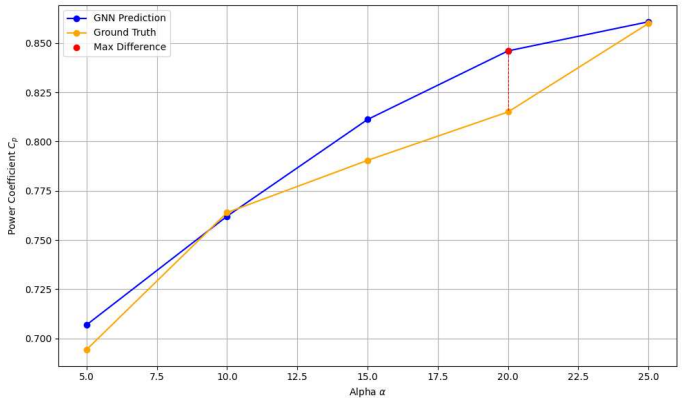


Figure 7.9: Ground truth C_p values for duct angles α ranging from $5°$ to $25°$, used as a reference dataset for optimization as compared with the C_p obtained by the GNN predicted flows.

in which v_∞ is the free-stream velocity of the wind upstream the turbine. The power coefficient is then given by

$$C_p = \frac{P}{P_{\text{wind}}}. \tag{7.16}$$

Using the Euler's equations for turbomachinery to calculate the power coefficient strikes a balance between computational efficiency and accuracy. By focusing on key flow components such as axial and tangential velocities, this approach avoids the complexity of methods like Blade Element Theory (BET), which require detailed geometric and aerodynamic modeling of each blade. The integration over the rotor radius further enhances accuracy by accounting for velocity variations across the span. This approach is particularly advantageous in optimization scenarios, where rapid evaluations of C_p are needed to explore large design spaces efficiently. This latter (Eq. 7.16) will eventually be the cost function used in the present work, with the aim to maximize it.
The ground truth data available for this work consist of power coefficient (C_p) values corresponding to specific duct angles α. The angles α range from $5°$ to $25°$ with a stride of $5°$. Although this dataset covers only a few angles, it serves as a proof of concept for validating the optimization process. The C_p values associated with these angles, for both the ground truth data and GNN predicted, are shown in Fig. 7.9, representing the performance of the wind turbine for these specific angles. The maximum of the discrepancy is observed at $\alpha = 20°$, where it reaches a value of 3.8%, which is considered acceptable for engineering design purposes. Although this discrepancy appears small in absolute terms, it still requires further investigation, as it reflects the limitations in the generalization capabilities of the GNN. In this case, we observe a monotonically increasing trend in the C_p values as the duct angle α increases.

7.6 . Reinforcement Learning (RL) in the Optimization Loop

The application of RL algorithms for optimization of fluid flows is a recent yet rapidly evolving area of research, showcasing significant potential for solving complex, high-dimensional problems. Several works have explored the application of RL in different flow control contexts. For example, Rabault et al. [2019] demonstrated the use of Proximal Policy Optimization (PPO), an actor-based RL method, to control the flow past a cylinder, achieving effective performance by leveraging the system's cumulative rewards. Similarly, Bucci et al. [2019, 2022] applied the Deep Deterministic Policy Gradient (DDPG) [Silver et al., 2014, Lillicrap et al., 2015] algorithm to stabilize nonlinear dynamics, such as those governed by the Kuramoto-Sivashinsky equation. Beyond flow control, RL has been applied to shape optimization problems. Notable works in this area include those by Viquerat et al. [2021], Keramati et al. [2022], Dussauge et al. [2023], which demonstrate the ability of RL to optimize shapes for improved system performance. Furthermore, adaptive mesh refinement using RL has gained traction in recent years, with innovative approaches proposed by Yang et al. [2023], Foucart et al. [2023] to dynamically refine meshes and enhance simulation accuracy. A recent review by Vignon et al. [2023] provides a comprehensive overview of these advancements, highlighting the growing importance of RL in fluid mechanics.

The goal of this section is not to provide an exhaustive introduction to RL, as the field is vast, multidisciplinary, and mathematically involved. Indeed, RL encompasses a wide range of methods, with deep mathematical foundations that extend beyond the scope of this work. For a more detailed exploration of RL, readers are referred to foundational resources such as Sutton and Barto [2018]. In this thesis, RL is used as a practical optimization tool. We adopt RL as a "plug-and-play" strategy, focusing on the DDPG algorithm [Silver et al., 2014, Lillicrap et al., 2015], which is well-suited for continuous action spaces like those found in fluid mechanics.

7.6.1 . Introduction to Reinforcement Learning (RL)

Reinforcement learning is a subfield of Artificial Intelligence (AI) in which an *agent* interacts with an environment to achieve a certain goal. The agent learns to make a sequence of decisions by taking actions in the environment, observing the results (*rewards*), and improving its decision-making *policy* over time. The ultimate goal is to find the optimal policy that maximizes the reward over time, evaluated by a *critic*. At a high level, RL involves the following core components:

- Agent: The agent is the decision-maker. It interacts with the environment by observing its current state s_t and selecting actions a_t to influence the environment's future states. The agent's objective is to maxi-

mize its cumulative reward R_t over time, which is achieved by improving its decision-making policy π through trial and error. In this work, the agent's role is to determine the optimal adjustments to the wind turbine's duct angle α in order to maximize the power coefficient (C_p).

- Environment: The environment represents the external system with which the agent interacts. It defines the state space, the available actions, and the rewards the agent receives as feedback. The environment in RL is dynamic, changing in response to the agent's actions. In our case, the environment is the wind turbine system, including the aerodynamic and physical characteristics that influence the power output as a result of changes in duct angle. The environment provides feedback to the agent based on how well the turbine performs with respect to energy extraction after each adjustment.

- State: The state s_t represents the environment at a specific time step t. It describes the current situation that the agent perceives, and it may include all or only a portion of the information necessary for decision-making. The state space could be continuous or discrete, and the agent must learn to take appropriate actions based on the current state. In this work, the state represents the current duct angle α.

- Action: The action a_t is the decision made by the agent at a given time step t. It represents the specific operation that influences the environment. The set of possible actions available to the agent is known as the action space, which can be either discrete or continuous. In a continuous action space, like in this work, the agent selects actions that gradually adjust the duct angle α to optimize C_p.

- Policy: The policy π is a strategy or set of rules that define how the agent selects actions based on the current state. In mathematical terms, a policy can be represented as

$$\pi : s_t \rightarrow a_t \tag{7.17}$$

  The policy is central to RL because it directly maps the information the agent observes about the environment (states, s_t) to its decisions (actions, a_t). As the agent learns, it refines its policy π to make better decisions and maximize rewards. In DDPG, a deterministic policy is used, meaning that for each observed state (e.g., a particular duct angle), the agent will select a specific adjustment action.

- Reward: The instantaneous reward r_t is a score associated with the action taken by the agent and the environment response to it. It is a scalar signal that indicates the success or failure of the action in terms

of achieving the agent's overall goal. The agent uses this feedback to assess the effectiveness of its actions and adjust its policy accordingly. The ultimate objective of the agent is to maximize the cumulative reward R_t it receives over time. Thus, the cumulative return R_t starting from a given time step t is defined as

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{7.18}$$

where γ is the discount factor $(0 \leq \gamma < 1)$ that determines the importance of future rewards. A higher γ places more emphasis on long-term rewards, while a lower γ makes the agent prioritize immediate rewards. In the context of optimizing the wind turbine, the reward is linked to the power coefficient (C_p): a higher C_p results in a higher reward, signaling to the agent that it has made a favorable adjustment to the duct angle.

- Value Function: The value function estimates the expected cumulative future reward from a given state-action pair. The value function can be thought of as a measure of long-term potential reward, taking into account not only immediate rewards but also the future rewards that the current state or action might lead to. There are two types of value functions commonly used in RL:

  - State Value Function ($V(s_t)$): It gives the expected cumulative reward starting from state s_t and following a certain policy thereafter:

    $$V(s_t) = \mathbb{E}[R_t|s_t] \tag{7.19}$$

  - Action Value Function or Q-Function ($Q(s_t, a_t)$): It gives the expected cumulative reward starting from state s_t, taking action a_t, and following the policy thereafter:

    $$Q(s_t, a_t) = \mathbb{E}[R_t|s_t, a_t] \tag{7.20}$$

- Critic: The critic is responsible for evaluating the actions taken by the agent based on the state-action pair. In DDPG, the critic approximates the Q-Function, which estimates the value of a particular action in a given state. This feedback helps the agent update its policy to choose actions that maximize long-term rewards. The critic uses the Bellman equation [Sutton and Barto, 2018] to iteratively improve its estimates of future rewards, learning how good an action is in terms of maximizing cumulative rewards. The critic and actor work together: the actor suggests actions, and the critic evaluates them, guiding the actor toward better policies.

This approach allows for control of complex, nonlinear systems through the interactions of the agent with the environment, assessed by the critic part. The following section will focus on how DDPG is applied to this problem to optimize the duct angle for maximizing the power coefficient (C_p).

7.6.2 . Application of DDPG in the Optimization Loop

The DDPG algorithm is an actor-critic method that is well-suited for optimization problems involving continuous control. In this work, we will use a continuous action space to achieve more precise optimization. By allowing the agent to explore continuous values, it can adjust the duct angle with fine precision to maximize the turbine's performance. In DDPG, the actor and the critic are represented by two NNs, namely two MLPs, to approximate key functions to guide the learning process.

The actor network is responsible for determining which action to take, given the current state of the system. In this context, the action represents an adjustment to the duct angle of the wind turbine. The critic network evaluates the quality of the actions taken by the actor. It does this by approximating the Q-Function, which predicts the expected cumulative reward (here, related to the power coefficient C_p) for a given state-action pair. The critic provides feedback to the actor, helping it improve its policy over time.

In RL, the agent interacts with the environment in episodes. During each episode, the agent observes the current state, takes an action (adjusts the duct angle), receives a reward (based on C_p), and transitions to a new state. The agent's goal is to maximize the cumulative reward over time by learning the best sequence of actions. To further stabilize learning, DDPG makes use of a *replay buffer*. During training, the agent stores past experiences in the buffer, where each experience consists of a tuple: state, action, reward, next state. Instead of updating the actor and critic after each individual experience, the agent samples mini-batches of experiences from the replay buffer. This random sampling breaks the temporal correlation between consecutive experiences, allowing the agent to learn more effectively from diverse situations.

In the context of this work, the state includes the current duct angle (and in future works possibly other environmental variables). The action is a continuous adjustment to the duct angle, and the reward is the power coefficient (C_p). The DDPG agent interacts with the wind turbine system over multiple episodes, continuously adjusting the duct angle based on the feedback it receives. As the agent observes the results of its actions, whether C_p increases or decreases, it learns to fine-tune its policy. Throughout the training process, the actor network improves its ability to output the best duct angle adjustments, while the critic network learns to evaluate these actions. Over time, this results in the agent converging to a policy that optimally adjusts the duct

angle, maximizing C_p and enhancing the overall performance of the wind turbine.

7.7 . The Mesh Generator

In this section, we describe the mesh generator script, which plays a crucial role in the optimization loop. It generates new mesh geometries based on the updated design parameters received from the reinforcement learning (RL) algorithm, specifically, the duct angle α in this case. This script is written in `Python` and uses the `Gmsh` [Geuzaine and Remacle, 2009] library to create and modify the mesh.
A critical feature of this mesh generation process is the refinement of the mesh in specific areas, particularly near the actuator disk. This local refinement is necessary to enhance the precision of the NN predictions, as accurate flow field predictions are vital in regions where energy extraction occurs. By increasing the density of mesh elements near the actuator disk, the generator ensures that the predicted flow characteristics are more detailed, which in turn improves the feedback provided to the RL agent during optimization.

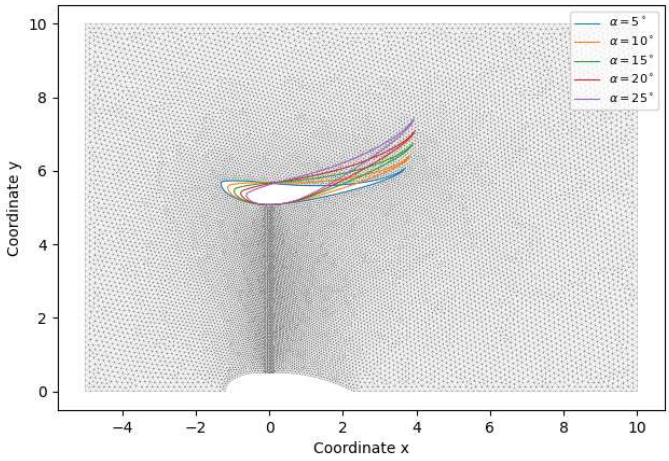


Figure 7.10: Cross-sectional generated mesh with refinement near the actuator disk at $x = 0$ along with the different duct positioning based on the angle α from $5°$ to $25°$

Fig. 7.10 illustrates a two-dimensional cross-sectional view of the generated wind turbine mesh. The most refined section of the mesh is located near the actuator disk, which is positioned at $x = 0$. This increased mesh density in the region surrounding the actuator disk ensures that the flow predictions in this critical area are highly accurate. In addition, the various duct positions

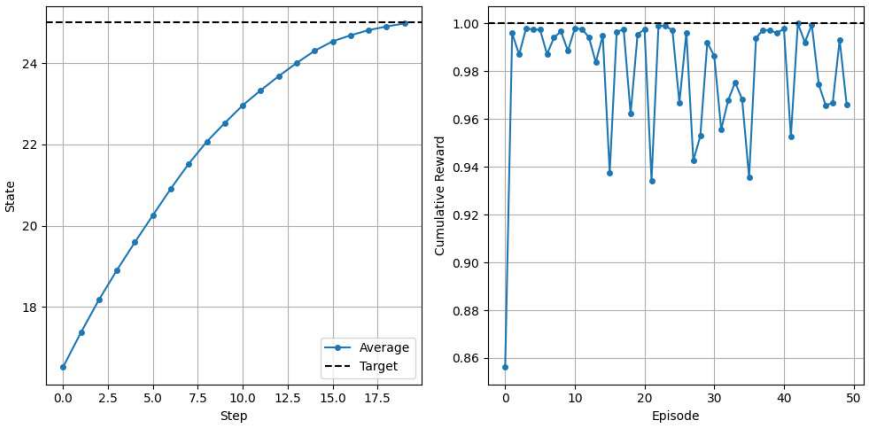


Figure 7.11: Optimization cycle performance: (Left) the optimization state curve as a function of the steps in the episodes (Right) the cumulative rewards per episode

corresponding to angles α ranging from $5°$ to $25°$ are overlaid in the figure. These ducts represent the different configurations evaluated in the current optimization process.
Fig. 7.11 shows how the RL-driven optimization follows a trajectory that matches the performance of the real system as represented by the ground truth. This demonstrates that the methodology is not only effective but also accurate, within the limits of the surrogate model used. The results obtained in this study are consistent with expectations, demonstrating that the optimization loop effectively finds the maximum power coefficient (C_p) within a small number of episodes. The iterative process of adjusting the duct angle, guided by the RL algorithm, shows that the system is capable of optimizing the C_p efficiently.

7.8 . Discussion

In this work we combined the use of a surrogate model based on GNN with an optimization process based on RL; in particular, by considering as optimization parameter the duct angle α, we demonstrated that it is possible to combine the two data-driven frameworks for obtaining robust prediction of the C_p (Fig. 7.9) and maximizing it. The performance of the optimization loop is effectively captured in Fig. 7.11, which illustrates both the evolution of the state s_t of the system over the step in each episode and the cumulative rewards R_t (Eq. 7.18) at each episode.

The left panel of Fig. 7.11 illustrates the optimization averaged state curve, where the state of the system s_t, specifically the duct angle α, as a function of the step within each episode. The curve presented is an average over all 50

episodes, providing a single view of how the state of the system s_t evolves throughout the optimization process. The curve shows how the RL agent gradually learns to adjust the duct angle α, with a consistent trend toward the maximum angle that optimizes C_p. As the episodes progress, α converges toward higher values, aligning with the physical understanding that increasing α maximizes the power coefficient C_p (Fig. 7.9).

The right panel of Fig. 7.11 presents the cumulative rewards R_t obtained during the training process. The cumulative reward reflects the agent's overall success in achieving its objective, in this case maximizing C_p. The initial cumulative rewards are relatively low, reflecting the agent's early phase of random exploration, where it lacks sufficient knowledge about the system to make effective adjustments to α. As training proceeds, the cumulative reward increases sharply, indicating rapid learning and an effective refinement of the agent's strategy π. This quick rise demonstrates that the agent is efficiently learning the relationship between the duct angle α and C_p. The steep increase followed by a stabilization at higher reward values confirms that the optimization problem is relatively simple, allowing the agent to learn an effective policy rapidly.

Additionally, the cumulative reward curve eventually oscillates around a maximum value. This behavior is expected in RL applications, particularly in scenarios where the environment presents some degree of variability or when the agent reaches a near-optimal policy. The oscillations indicate that the agent has found a high-performing strategy but continues to explore minor variations around this optimal configuration to fine-tune its performance. This exploration-exploitation balance is a distinctive feature of RL, as the agent seeks to maximize long-term rewards while still occasionally testing slight adjustments to ensure it is not missing any better configurations.

Notably, the GNN used in this study was not subject to any hyperparameter optimization (Sec. 3.8.3). The robustness of the learning framework is demonstrated by the fact that the same hyperparameters, which was optimized in a different case study (Sec. 5 or Sec. 6), also perform well in this entirely different application. This indicates the general effectiveness of the chosen GNN configuration. A hyperparameters optimization would likely enhance the GNN's predictive accuracy and the overall efficiency of the optimization loop.

While the study focuses on a relatively simple scenario due to the monotonic dependence of the coefficient C_p as a function of α, the primary aim is to demonstrate the methodology and the ability of the RL algorithm, combined with the NN surrogate model, to optimize the duct geometry effectively.

In future works, we aim to expand this optimization framework to address more complex and realistic scenarios. Specifically, the approach will be extended to include additional geometric design variables and control param-

eters that influence the resulting flow field. By broadening the range of variables under consideration, we will aim at optimizing both the geometry and the flow characteristics simultaneously. As already pointed out in Sec. 7.1, the optimization framework will be adapted to accommodate multi-objective optimization problems, balancing aerodynamic efficiency with other critical factors, such as flow regularity, structural integrity, or environmental impact. The RL framework is deemed suitable to manage large and complex optimization spaces, like the ones represented by parametrized turbine designs. This generalization will be crucial for applications such as wind farm optimization, where minimizing downstream turbulence and maximizing overall energy extraction are essential, or urban wind turbine setups, where minimizing the disturbance to the environment is a key objective. To this end, future work will also focus on enhancing the GNN's capability to generalize across a broader range of geometries and flow conditions, which will lead to even more accurate and efficient optimizations.

8 - Conclusion

The primary objective of this thesis was to investigate the integration of a ML algorithm, specifically Graph Neural Network (GNN)s, into the field of Computational Fluid Dynamics (CFD), focusing on data assimilation problems and optimization of fluid systems. This research is driven by the potential of ML tools to address some of the limitations of traditional CFD methods, particularly their high computational cost. These challenges often stem from the complex parameterization of fluid systems and the need to analyze various boundary conditions, flow regimes, and geometric configurations, which typically require extensive additional CFD simulations. By incorporating GNNs, we aimed to explore practical ways to face these challenges, ultimately supporting the cited aspects of fluid dynamics workflows. This integration is significant as emerging computational tools, such as ML, and the increasing of computational power are becoming available to support and enhance traditional CFD techniques.

Our research was organized around three main subjects. The first one was to address the RANS closure problem by determining whether GNNs could effectively learn the closure terms for (RANS) models, which can be challenging to determine when complex geometries are analyzed or the access to full data is limited. Traditional RANS models depend significantly on manually tuned turbulence closure models, which are inherently empirical and often lack generalizability across different flow conditions. By leveraging high-fidelity Direct Numerical Simulation (DNS) data, we demonstrated that GNNs could potentially learn these closure terms directly and reduce reliance on traditional turbulence models in certain cases. It is important to emphasize that the aim here is not to replace traditional turbulence modeling approaches, but rather to demonstrate how innovative tools like GNNs can provide more adaptive and general frameworks. Additionally, we employed an active learning strategy to efficiently select data for the training dataset, ensuring that only the most informative data points were included. This approach minimized the inclusion of redundant data and enhanced the GNN's performance and accuracy.

Secondly, we explored the importance of physical consistency in ML models. While data-driven techniques have proven powerful, they often produce solutions that may not fully adhere to fundamental physical laws, limiting their application to relevant engineering problems. In response, we embedded physical constraints into the GNN training process. This physics-constrained approach helps ensure that the network's outputs are not only statistically accurate but also consistent with core fluid mechanics principles. This methodology aims to bridge the gap between purely empirical modeling and phys-

ically grounded predictions, thereby improving the model's robustness and fidelity for engineering applications. Although incorporating such physical constraints shows promises, it remains an area that requires further refinement, as this work primarily aims to show a possible technique to include the physics of the problem into the GNN training process. ML remains fundamentally a statistical tool, contrasting with the deterministic nature of traditional CFD methods, and therefore cannot be fully relied upon on its own. This part of the thesis aims at demonstrating, once again, how this approach can potentially support deterministic CFD techniques, and how a combination of the two might be both efficient and productive.

The third subject we addressed was the optimization of fluid systems, with a focus on practical engineering applications. We employed GNNs as surrogate models to optimize the design of Diffuser-Augmented Wind Turbine (DAWT) ducts, with the aim to significantly reduce the computational cost associated with the CFD simulations running during the iterative optimization process for the cost function evaluations. The approach enabled a more efficient exploration of the design space, serving as a proof of concept for using GNNs in optimization environments. The results suggest that a surrogate ML model can be used in fluid dynamics optimization, enabling faster convergence to workable designs and providing a starting point for more comprehensive future studies. Even though the application here is limited in terms of the number of parameters involved in the optimization process, this effort serves primarily as a preliminary demonstration of the potential for efficient numerical optimizations. More complex optimizations will require further development and testing to establish their effectiveness.

Looking ahead, several promising avenues for further research await exploration. While GNN-based models have shown potential, further refinements are necessary. From the training process viewpoint, incorporating more complex and diverse datasets, as well as exploring additional neural network architectures, could enhance the performance and the generalizability of the models. For example, hybrid architectures that combine GNNs with Recurrant Neural Networks (RNN)s or transformers may prove useful in adapting to various flow conditions and improving predictive accuracy. Additionally, the development of adaptive learning methods, such as active learning, could optimize the training process by selectively focusing on the most informative data. This targeted approach to data usage has the potential to reduce training times and enhance the overall robustness and efficiency of the models.

Another area for future exploration is the integration of uncertainty quantification into our GNN framework. As ML models are increasingly deployed in engineering systems, understanding and mitigating prediction uncertainties becomes crucial. Incorporating methods for uncertainty quantification would improve the reliability of GNN-based models and support decision-making

processes in tasks such as design optimization.

Furthermore, applying these methodologies to more complex flow scenarios, such as three-dimensional flows or fully turbulent flows, could help demonstrate the potential utility of GNNs as these are indeed the scenarios where traditional CFD approaches face significant computational challenges and could benefit most from additional support.

Lastly, the combination of Reinforcement Learning (RL) and fluid dynamics optimization holds some promises. The preliminary results reported in the thesis enter an already wide literature on the potential for reinforcement learning to effectively optimize dynamic systems. RL may prove particularly useful in applications requiring continual adaptation, such as the control of unsteady aerodynamic surfaces, adaptive turbine blade adjustments, or the real-time optimization fluid systems. By integrating RL with GNN-based models, it may be possible to create systems capable of learning and adapting in real-time, potentially improving operational efficiency in specific applications.

In conclusion, this thesis has aimed to provide a contribution towards integrating GNNs into CFD workflows, demonstrating some predictive and optimization capabilities. The findings presented here suggest potential directions for broader applications that could contribute to more efficient fluid dynamics approaches. As ML technologies continue to evolve, its role in fluid mechanics may grow, offering useful tools to face certain engineering challenges. While GNNs show promise, significant challenges remain, and continued research and refinement are necessary to fully understand and utilize their potential. The work presented in this thesis should be seen as an exploratory step in integrating modern ML tools with traditional fluid dynamics approaches, providing insights that could inform both engineering and scientific research in the future.

9 - Acknowledgements

Here we are, at the end of this extraordinary three-year journey. It's been an unforgettable experience, a whirlwind of challenges and triumphs, and above all, an emotional rollercoaster that has left an indelible mark on me. As I reflect on this chapter of my life, I'm overwhelmed with gratitude for the incredible people who made it all possible.

First and foremost, I want to express gratitude to myself. With the risk to sound selfish, but I believe in acknowledging the perseverance and courage it took to navigate this journey. The person who entered the PhD program three years ago is almost unrecognizable compared to the one writing these words. I faced some of the hardest moments of my life during this time, but I emerged stronger, more resilient, and more self-aware. So, thank you, Michele, for not giving up, for adapting, and for growing. You did well.

This journey would not have been possible without the invaluable guidance and mentorship of my thesis directors, Prof. Stefania Cherubini and Prof. Caroline Nore, and my co-supervisors, Dr. Onofrio Semeraro and Dr. Michele Alessandro Bucci. Your expertise, patience, and unwavering support were my anchors throughout this process. A special thank you goes to Onofrio, who has been a constant presence from day one, offering not only his professional insight but also his encouragement. Often, it felt like we were walking this path together.

To my family — my parents and my two brothers — your love and support have been the firm foundation I could always rely on. No matter how tumultuous things became, you were always there, ready to lift me up and remind me of my strength. Your unwavering belief in me was my greatest source of motivation.

To Annalisa, you were my rock during the final and most intense moments of this journey. Your patience, understanding, and ability to weather the storm of emotions with me have been nothing short of extraordinary. Thank you for standing by my side.

To my colleagues and friends, who turned what could have been a solitary journey into a shared adventure: Amine Saibi, Thibault Monsel, Remy Hosseinkhan — I'm proud to call you friends. Your camaraderie and laughter brought light to the darkest days. To Cyril, Stephan, Emanuel, Matthieu, Arthur, Alice, and so many others, thank you for your collaboration and support. Each of you has left a unique imprint on my experience, and I will carry those memories with me as I step into the next chapter.

To Matteo, Anna, Ivan, Mary, Bianca (and Martino and Irene), the Italian Team in Paris, thank you for being my family away from home. With you, I shared a journey full of moments that will stay with me rent-free: the chal-

lenges we faced together (from the COVID19), the endless laughter, the adventures that took us to new places (Normandy, Bordeaux), and the celebrations that marked our triumphs. Your presence turned this experience into something deeply meaningful, and I hold you all close to my heart.

To the jury members, thank you for your time, your insightful questions, and your encouragement. It was an honor to engage with such esteemed researchers, and your recognition of my work is something I will always cherish.

Lastly, to everyone who believed in me and contributed to this journey in ways big and small: thank you. I'm filled with a sense of gratitude and pride as I close this chapter and prepare to embark on a new adventure in the "real world." Wherever this path takes me, I will carry your support and the lessons I've learned here in my heart.

Thank you, from the bottom of my heart.

List of Figures

List of Tables

Bibliography

T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd Association for Computing Machinery International Conference on Knowledge Discovery and Data Mining*, 2019.

M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E Rognes, and G. N Wells. The FEniCS project version 1.5. *Archive of Numerical Software*, 3(100), 2015.

O. Amoignon, J.O. Pralits, A. Hanifi, M. Berggren, and D. S. Henningson. Adjoint-based shape optimization for natural laminar flow design. In *ERCOFTAC Design Optimization: Methods & Applications Conference*, Athens, Greece, 2004.

R. Anatol. Naca report 1191. 1958.

ANSYS Inc. *ANSYS Fluent, Release 2023 R2*. ANSYS Inc., Canonsburg, PA, USA, 2023. Available at: https://www.ansys.com/products/fluids/ansys-fluent.

A. C. Aranake, V. K. Lakshminarayan, and K. Duraisamy. Computational analysis of shrouded wind turbine configurations using a 3-dimensional rans solver. *Renewable Energy*, 75:818–832, 2015.

A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115, 2020. ISSN 1566-2535. doi: https://doi.org/10.1016/j.inffus.2019.12.012. URL https://www.sciencedirect.com/science/article/pii/S1566253519308103.

A. Beck and M. Kurz. A perspective on machine learning methods in turbulence modeling. *GAMM-Mitteilungen*, 44(1):e202100002, 2021.

F. Belbute-Peres, D. T. Economon, and J. Z. Kolter. Combining differentiable pde solvers and graph neural networks for fluid flow prediction, 2020. URL https://arxiv.org/abs/2007.04439.

Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 41–48, 2009.

R. Bontempo and E. M. Di Marzo. Diffuser augmented wind turbines: A critical analysis of the design practice based on the ducting of an existing open rotor. *Journal of Wind Engineering and Industrial Aerodynamics*, 238:105428, 2023. doi: 10.1016/j.jweia.2023.105428.

R. Bontempo and M. Manna. The joukowsky rotor for diffuser augmented wind turbines: design and analysis. *Energy Conversion and Management*, 252:114952, 2022.

A. Borzi and V. Schulz. *Computational Optimization of Systems Governed by Partial Differential Equations*. SIAM, 2012.

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

S. L. Brunton, M. S. Hemati, and K. Taira. Special issue on machine learning and data-driven methods in fluid dynamics. *Theoretical and Computational Fluid Dynamics*, 34(4):333–337, 2020a.

S.L. Brunton, B.R. Noack, and P. Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020b.

M. A. Bucci, O. Semeraro, A. Allauzen, L. Cordier, and L. Mathelin. Nonlinear optimal control using deep reinforcement learning. In *International Conference on Machine Learning*, page Chapter 24, 2022. doi: 10.1007/978-3-030-67902-6_24.

M. A. Bucci, O. Semeraro, A. Allauzen, S. Chibbaro, and L. Mathelin. Curriculum learning for data-driven modeling of dynamical systems. *The European Physical Journal E*, 46(3):12, 2023.

M.A. Bucci, O. Semeraro, A. Allauzen, G. Wisniewski, L. Cordier, and L. Mathelin. Control of chaotic systems by deep reinforcement learning. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 475 (2231):20190351, 2019.

T. Burton, D. Sharpe, N. Jenkins, and E. Bossanyi. *Wind Energy Handbook*. John Wiley & Sons, 2nd edition, 2011.

K. T. Butler, D. W. Davies, H. Cartwright, O. Isayev, and A. Walsh. Machine learning for molecular and materials science. *Nature*, 559:547–555, 2018.

J. Cai, P.E. Angeli, J.M. Martinez, G. Damblin, and D. Lucor. Revisiting tensor basis neural network for reynolds stress modeling: Application to plane channel and square duct flows. *Computers & Fluids*, page 106246, 2024.

S. Cai, Z. Mao, and Z. et al. Wang. Physics-informed neural networks (pinns) for fluid mechanics: a review. *Acta Mechanica Sinica*, 2021.

A. S. Cato, P. S. Volpiani, V. Mons, O. Marquet, and D. Sipp. Comparison of different data-assimilation approaches to augment rans turbulence models. *Computers & Fluids*, 266:106054, 2023. doi: 10.1016/j.compfluid.2023.106054.

G. Charpiat, N. Girard, L. Felardos, and Y. Tarabalka. Input similarity from the neural network perspective. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/c61f571dbd2fb949d3fe5ae1608dd48b-Paper.pdf.

F. Charru. *Hydrodynamic Instabilities*, volume 37. Cambridge University Press, 2011.

J. Chen, E. Hachem, and J. Viquerat. Graph neural networks for laminar flow prediction around random two-dimensional shapes. *Physics of Fluids*, 33 (12):123607, 2021.

S. Cherubini, P. De Palma, J. Ch. Robinet, and A. Bottaro. Rapid path to transition via nonlinear localized optimal perturbations in a boundary-layer flow. *Phys. Rev. E*, 82:066302, December 2010. doi: 10.1103/PhysRevE.82.066302. URL https://link.aps.org/doi/10.1103/PhysRevE.82.066302.

S. Cherubini, J. C. Robinet, and P. De Palma. Nonlinear control of unsteady finite-amplitude perturbations in the blasius boundary-layer flow. *Journal of Fluid Mechanics*, 737:440–465, 2013. doi: 10.1017/jfm.2013.576.

A. Chiarini, M. Quadrio, and F. Auteri. Linear stability of the steady flow past rectangular cylinders. *Journal of Fluid Mechanics*, 929, December 2021. doi: 10.1017/jfm.2021.819.

A. Chiarini, M. Quadrio, and F. Auteri. A new scaling for the flow instability past symmetric bluff bodies. *Journal of Fluid Mechanics*, 936:R2, 2022. doi: 10.1017/jfm.2022.99.

S. Colabrese, K. Gustavsson, A. Celani, and L. Biferale. Flow navigation by smart microswimmers via reinforcement learning. *Physical Review Letters*, 118:158004, 2017. doi: 10.1103/PhysRevLett.118.158004.

B. Colvert, M. Alsalman, and E. Kanso. Classifying vortex wakes using neural networks. *Bioinspiration and Biomimetics*, 13(2):025003, 2018.

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3): 273–297, 1995.

S. Costanzo, T. Sayadi, M. Fosas de Pando, P.J. Schmid, and P. Frey. Parallel-in-time adjoint-based optimization–application to unsteady incompressible flows. *Journal of Computational Physics*, 471:111664, 2022.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

G. De Cillis, S. Cherubini, O. Semeraro, S. Leonardi, and P. De Palma. The influence of incoming turbulence on the dynamic modes of an nrel-5mw wind turbine wake. *Renewable Energy*, 183:601–616, 2022. doi: https://dx.doi.org/10.1016/j.renene.2021.11.037.

A. Dilimulati, T. Stathopoulos, and M. Paraschivoiu. Wind turbine designs for urban applications: A case study of shrouded diffuser casing for turbines. *Journal of Wind Engineering and Industrial Aerodynamics*, 175:179–192, 2018.

B. Donon, Z. Liu, W. Liu, I. Guyon, A. Marot, and M. Schoenauer. Deep statistical solvers. *Advances in Neural Information Processing Systems*, 33:7910–7921, 2020.

E. Dow and Q. Wang. Output based dimensionality reduction of geometric variability in compressor blades. In *Proceedings of the 51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, Grapevine, TX, 2013. AIAA Paper 2013-0420.

P.G. Drazin and W.H. Reid. *Hydrodynamic Instabilities*. Cambridge University Press, 2nd edition, 2002. ISBN 9780521538880.

D. Dupuy, N. Odier, and C. Lapeyre. Data-driven wall modeling for turbulent separated flows. *Journal of Computational Physics*, 487:112173, 2023a. doi: 10.1016/j.jcp.2023.112173.

D. Dupuy, N. Odier, C. Lapeyre, and D. Papadogiannis. Modeling the wall shear stress in large-eddy simulation using graph neural networks. *Data-Centric Engineering*, 4:e7, 2023b.

K. Duraisamy, G. Iaccarino, and H. Xiao. Turbulence modeling in the age of data. *Annual Review of Fluid Mechanics*, 51:357–377, 2019.

T. P. Dussauge, W. J. Sung, Olivier J. P. F., and D. N. Mavris. A reinforcement learning approach to airfoil shape optimization. *Scientific Reports*, 13(1):9753, 2023.

H. Eivazi, M. Tahani, P. Schlatter, and R. Vinuesa. Physics-informed neural networks for solving reynolds-averaged navier–stokes equations. *Physics of Fluids*, 34(7):075117, 2022.

A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. S. Corrado, S. Thrun, and J. Dean. A guide to deep learning in healthcare. *Nature Medicine*, 25(1):24–29, 2019.

S. Etienne and D. Pelletier. Effect of rotational degree of freedom on vortex-induced vibrations of a circular cylinder in cross-flow. *Journal of Fluids and Structures*, 57:1–17, 2015.

M. Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

K. M. Foreman, B. Gilbert, and R. A. Oman. Diffuser augmentation of wind turbines. *Solar Energy*, 20(4):305–311, 1978.

C. Foucart, A. Charous, and P. F. Lermusiaux. Deep reinforcement learning for adaptive mesh refinement. *Journal of Computational Physics*, 491:112381, 2023.

D. P. G. Foures, N. Dovetta, D. Sipp, and P. J. Schmid. A data-assimilation method for Reynolds-averaged Navier–Stokes-driven mean flow reconstruction. *Journal of Fluid Mechanics*, 759:404–431, 2014.

K. Fukami, K. Fukagata, and K. Taira. Super-resolution reconstruction of turbulent flows with machine learning. *arXiv preprint arXiv:1811.11328*, 2018.

P. Garnier, J. Viquerat, J. Rabault, A. Larcher, A. Kuhnle, and E. Hachem. A review on deep reinforcement learning for fluid mechanics. *Computers & Fluids*, 225:104973, 2021.

C. Geuzaine and J.F. Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009. doi: $10.1002/\mathrm{nme}.2579$.

F. Giannetti and P. Luchini. Structural sensitivity of the first instability of the cylinder wake. *Journal of Fluid Mechanics*, 581:167–197, 2007.

M. B. Giles and N. A. Pierce. An introduction to the adjoint approach to design. *Flow, Turbulence and Combustion*, 65(3):393–415, 2000.

D. E. Goldberg and J. H. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3:95–99, 1988. doi: $10.1023/\mathrm{A}{:}1022602019183$.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

T. Guégan. *Contrôle d'écouleements turbulents par apprentissage automatique*. Université de Poitiers, 2022.

S. Hassanli, K. Chauhan, M. Zhao, and K.C.S. Kwok. Application of through-building openings for wind energy harvesting in built environment. *Journal of Wind Engineering and Industrial Aerodynamics*, 184:445–455, 2019.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

M. Hinze, R. Pinnau, M. Ulbrich, and S. Ulbrich. *Optimization with PDE Constraints*. Springer, 2008.

J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79 (8):2554–2558, 1982.

K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

A. Ilhan, B. Sahin, and M. Bilgili. A review: Diffuser augmented wind turbine technologies. *International Journal of Green Energy*, 19(1):1–27, 2021.

A. Jameson. Aerodynamic design via control theory. *Journal of Scientific Computing*, 3(3):233–260, 1988.

A. Jameson. Optimum aerodynamic design using cfd and control theory. In *37th Aerospace Sciences Meeting and Exhibit*. American Institute of Aeronautics and Astronautics, 1995.

M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

E. Kaiser, B.R. Noack, L. Cordier, A. Spohn, M. Segond, M. Abel, G. Daviller, G. Tadmor, and R.K. Niven. Cluster-based reduced-order modelling of a mixing layer. *Journal of Fluid Mechanics*, 754:365–414, 2014.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995. doi: 10.1109/ICNN.1995.488968.

H. Keramati, F. Hamdullahpur, and M. Barzegari. Deep reinforcement learning for heat exchanger shape optimization. *International Journal of Heat and Mass Transfer*, 194:123112, 2022.

S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. doi: 10.1126/science.220.4598.671. URL https://www.science.org/doi/abs/10.1126/science.220.4598.671.

C. J. Lapeyre, A. Misdariis, N. Cazard, D. Veynante, and T. Poinsot. Training convolutional neural networks to estimate turbulent sub-grid scale reaction rates. *Combustion and Flame*, 203:255–264, 2019.

B.E. Launder and B.I. Sharma. Application of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc. *Letters in Heat and Mass Transfer*, 1(2):131–137, 1974.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

C. Lee, J. Kim, D. Babcock, and R. Goodman. Application of neural networks to turbulence control for drag reduction. *Physics of Fluids*, 9(6):1740–1747, 1997. doi: $10.1063/1.869290$.

S. Lee and D. You. Data-driven prediction of unsteady flow over a circular cylinder using deep learning. *Journal of Fluid Mechanics*, 879:217–254, 2019.

S. Li, J. Wang, and S. Cai. Numerical simulation of flow past a circular cylinder at different reynolds numbers. *Journal of Hydrodynamics*, 17(3):326–332, 2005.

M. W. Libbrecht and W. S. Noble. Machine learning applications in genetics and genomics. *Nature Reviews Genetics*, 16:321–332, 2015.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

J. Ling and J. Templeton. Evaluation of machine learning algorithms for prediction of regions of high Reynolds-averaged Navier Stokes uncertainty. *Physics of Fluids*, 27(8):085103, 2015.

J. Ling, A. Kurzawski, and J. Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166, 2016.

M. Lino, S. Fotiadis, A. A. Bharath, and C. D. Cantwell. Current and emerging deep-learning methods for the simulation of fluid dynamics. *Proceedings of the Royal Society A*, 479(2275):20230058, 2023.

J.L. Lions. *Optimal Control of Systems Governed by Partial Differential Equations*. Springer-Verlag, 1971.

C. Liu, X. Zheng, and H.J. Sung. Numerical simulations of two-dimensional flows past a circular cylinder. *Journal of Fluids and Structures*, 12(7):851–856, 1998.

A. Logg, K.A. Mardal, and G. Wells. *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*, volume 84 of *Lecture Notes in Computational Science and Engineering*. Springer-Verlag, 2012. ISBN 978-3-642-23098-1. doi: 10.1007/978-3-642-23099-8.

J.C. Loiseau, J.C. Robinet, S. Cherubini, and E. Leriche. Investigation of the roughness-induced transition: global stability analyses and direct numerical simulations. *Journal of Fluid Mechanics*, 760:175–211, 2014. doi: 10.1017/jfm.2014.589.

P. Luchini and A. Bottaro. Adjoint equations in stability analysis. *Annual Review of Fluid Mechanics*, 46(Volume 46, 2014):493–517, 2014. ISSN 1545-4479. doi: https://doi.org/10.1146/annurev-fluid-010313-141253. URL https://www.annualreviews.org/content/journals/10.1146/annurev-fluid-010313-141253.

P. Luchini, F. Giannetti, and J.O. Pralits. Structural sensitivity of the finite-amplitude vortex shedding behind a circular cylinder. In *IUTAM Symposium on Unsteady Separated Flows and their Control*, pages 151–160. Springer, 2009.

O. Marquet, D. Sipp, and L. Jacquin. Sensitivity analysis and passive control of cylinder flow. *Journal of Fluid Mechanics*, 615:221–252, 2008. doi: 10.1017/S0022112008003662.

R. McConkey, E. Yee, and F.S. Lien. On the generalizability of machine-learning-assisted anisotropy mappings for predictive turbulence modelling. *International Journal of Computational Fluid Dynamics*, 36(7):555–577, 2022.

M. A. Mendez, M. Balabane, and J.-M. Buchlin. Multi-scale proper orthogonal decomposition of complex fluid flows. *Journal of Fluid Mechanics*, 870: 988–1036, May 2019. ISSN 1469-7645. doi: 10.1017/jfm.2019.212. URL http://dx.doi.org/10.1017/jfm.2019.212.

M. A. Mendez, J. Dominique, M. Fiore, F. Pino, P. Sperotto, and J. Van den Berghe. Challenges and opportunities for machine learning in fluid mechanics. *arXiv preprint arXiv:2202.12577*, 2022.

M. A. Mendez, A. Ianiro, B. R. Noack, and S. L. Brunton. *Data-Driven Fluid Mechanics: Combining First Principles and Machine Learning*. Cambridge University Press, Cambridge, UK, 2023. ISBN 978-1-108-84214-3.

F. R. Menter, M. Kuntz, and R. Langtry. Ten years of industrial experience with the sst turbulence model. *Turbulence, Heat and Mass Transfer*, 4(1):625–632, 2003.

M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, 1969.

P. Moin and K. Mahesh. Direct numerical simulation: A tool in turbulence research. *Annual Review of Fluid Mechanics*, 30:539–578, 1998.

P. Morra, O. Semeraro, D.S. Henningson, and C. Cossu. On the relevance of reynolds stresses in resolvent analyses of turbulent wall-bounded flows. *Journal of Fluid Mechanics*, 867:969–984, 2019.

M. Nastorg. *Scalable GNN Solutions for CFD Simulations*. Thesis, Université Paris-Saclay, April 2024. URL https://theses.hal.science/tel-04590477.

J.E. Parish and K. Duraisamy. A paradigm for data-driven predictive modeling using field inversion and machine learning. *Journal of Computational Physics*, 305:758–774, 2016. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2015.11.012. URL https://www.sciencedirect.com/science/article/pii/S0021999115007524.

M. A. Park. Adjoint-based, three-dimensional error prediction and grid adaptation. In *Proceedings of the 32nd AIAA Fluid Dynamics Conference and Exhibit*, St. Louis, MO, 2002. AIAA. AIAA Paper 2002-3286.

Y. Patel, V. Mons, O. Marquet, and G. Rigas. Turbulence model augmented physics-informed neural networks for mean-flow reconstruction. *Physical Review Fluids*, 9(3):034605, 2024.

T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2021.

E. Pickering, G. Rigas, O.T. Schmidt, D. Sipp, and T. Colonius. Optimal eddy viscosity for resolvent-based models of coherent structures in turbulent jets. *Journal of Fluid Mechanics*, 917:A29, 2021.

F. Pino, L. Schena, J. Rabault, A. Kuhnle, and M. A. Mendez. Comparative analysis of machine learning methods for active flow control. *Journal of Fluid Mechanics*, 958:A2, 2023.

O. Pironneau. On optimum design in fluid mechanics. *Journal of Fluid Mechanics*, 64(1):97–110, 1974.

S. B. Pope. A more general effective-viscosity hypothesis. *Journal of Fluid Mechanics*, 72(2):331–340, 1975.

S. B. Pope. Turbulent flows. *Cambridge University Press*, 2000.

F Porté-Agel, M. Bastankhah, and S. Shamsoddin. Wind-turbine and wind-farm flows: A review. *Boundary layer metheorology*, 59, 2020.

T. Potsis, Y. Tominaga, and T. Stathopoulos. Computational wind engineering: 30 years of research progress in building structures and environment. *Journal of Wind Engineering and Industrial Aerodynamics*, 234:105346, 2023. doi: 10.1016/j.jweia.2023.105346.

M. Provansal, C. Mathis, and L. Boyer. Benard-von Karman instability: transient and forced regimes. *Journal of Fluid Mechanics*, 182:1–22, 1987. doi: 10.1017/S0022112087002222.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

J. Rabault, M. Kuchta, A. Jensen, U. Réglade, and N. Cerardi. Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *Journal of Fluid Mechanics*, 865:281–302, 2019.

A. Radovic, M. Williams, D. Rousseau, M. Kagan, D. Bonacorsi, A. Himmel, S. Ahuja, and D. Trocino. Machine learning at the energy and intensity frontiers of particle physics. *Nature*, 560:41–48, 2018.

M. Raissi, A. Yazdani, and G.E. Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481): 1026–1030, 2020.

S. S. Rao. *Engineering Optimization: Theory and Practice*. John Wiley & Sons, 2009.

P. Ren, Y. Xiao, X. Chang, P.Y. Huang, Z. Li, B. B. Gupta, X. Chen, and X. Wang. A survey of deep active learning. *Association for Computing Machinery computing surveys*, 54(9):1–40, 2021.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

C. L. Rumsey and S. X. Ying. Prediction of high lift: review of present cfd capability. *Progress in Aerospace Sciences*, 38:145–180, 2002.

A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pages 4470–4479, 2018.

F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

M. Schmelzer, R. P. Dwight, and P. Cinnella. Discovery of algebraic reynolds-stress models using sparse symbolic regression. *Flow, Turbulence and Combustion*, 104:579–603, 2020.

O. Semeraro, J. O. Pralits, C. W. Rowley, and D. S. Henningson. Riccati-less approach for optimal control and estimation: an application to two-dimensional boundary layers. *Journal of Fluid Mechanics*, 731:394–417, 2013. doi: 10.1017/jfm.2013.352.

K. Shukla, M. Xu, N. Trask, and G. E. Karniadakis. Scalable algorithms for physics-informed neural and graph networks. *Data-Centric Engineering*, 3, 2022.

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, pages 387–395. PMLR, 2014.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

A. P. Singh and K. Duraisamy. Using field inversion to quantify functional errors in turbulence closures. *Physics of Fluids*, 28(4):045110, 2016.

T. Stathopoulos, H. Alrawashdeh, A. Al-Quraan, B. Blocken, A. Dilimulati, M. Paraschivoiu, and P. Pilay. Urban wind energy: Some views on potential and challenges. *Journal of Wind Engineering and Industrial Aerodynamics*, 179: 146–157, 2018.

C. A. Ströfer and H. Xiao. End-to-end differentiable learning of turbulence models from indirect observations. *Theoretical and Applied Mechanics Letters*, 11(4):100280, 2021.

L. Sun, J. Zhang, L. Schaefer, and Q. Wang. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering*, 361:112732, 2020.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

S. Thrun and L. Pratt. Learning to learn: Introduction and overview. In *Learning to Learn*, pages 3–17. Springer, 1998.

N. Thuerey, K. Weißenow, L. Prantl, and X. Hu. Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal*, 58(1):25–36, 2020.

A. P. Toshev, G. Galletti, J. Brandstetter, S. Adami, and N. A. Adams. Learning lagrangian fluid mechanics with e (3)-equivariant graph neural networks. In *International Conference on Geometric Science of Information*, pages 332–341. Springer, 2023.

A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

G.J.W. van Bussel. The science of making more torque from wind: Diffuser experiments and theory revisited. *Journal of Physics: Conference Series*, 75 (1):012010, 2007.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, volume 30, pages 5998–6008, 2017.

D. A. Venditti and D. L. Darmofal. Anisotropic grid adaptation for functional outputs: application to two-dimensional viscous flows. *Journal of Computational Physics*, 187:22–46, 2003.

R. Venters, B. T. Helenbrook, and K. D. Visser. Ducted wind turbine optimization. In *34th AIAA Applied Aerodynamics Conference*, 2016. doi: 10.2514/6.2016-3729.

C. Vignon, J. Rabault, and R. Vinuesa. Recent advances in applying deep reinforcement learning for flow control: Perspectives and future directions. *Physics of Fluids*, 35(3), 2023.

R. Vinuesa and S. L. Brunton. Enhancing computational fluid dynamics with machine learning. *Nature Computational Science*, 2(6):358–366, 2022.

J. Viquerat, J. Rabault, A. Kuhnle, H. Ghraieb, A. Larcher, and E. Hachem. Direct shape optimization through deep reinforcement learning. *Journal of Computational Physics*, 428:110080, 2021.

P.S. Volpiani, M. Meyer, L. Franceschini, J. Dandois, F. Renac, E. Martin, O. Marquet, and D. Sipp. Machine learning-augmented turbulence modeling for RANS simulations of massively separated flows. *Physical Review Fluids*, 6(6): 064607, 2021.

J. G. von Saldern, J. M. Reumschüssel, T. L. Kaiser, O. T. Schmidt, P. Jordan, and K. Oberleithner. Self-consistent closure modeling for linearized mean field methods. In *AIAA AVIATION 2023 Forum*, page 4351, 2023.

J.X. Wang, J.L. Wu, and H. Xiao. Physics-informed machine learning approach for reconstructing reynolds stress modeling discrepancies based on dns data. *Physical Review Fluids*, 2(3):034603, 2017.

J.X. Wang, J. Wu, and H. Xiao. Incorporating prior knowledge for turbulence modeling with data-driven approaches: Physics-informed machine learning for the rans closure problem. *Physics of Fluids*, 32(3):035116, 2020.

Q. Wang, F. Ham, G. Iaccarino, and P. Moin. Risk quantification in unsteady flow simulations using adjoint-based approaches. In *Proceedings of the 50th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Palm Springs, CA, 2009. AIAA Paper 2009-2277.

J. Weatheritt and R. Sandberg. A novel evolutionary algorithm applied to algebraic modifications of the RANS stress–strain relationship. *Journal of Computational Physics*, 325:22–37, 2016.

F. M. White. *Viscous Fluid Flow*. McGraw-Hill, 2006.

D. C. Wilcox. *Turbulence Modeling for CFD*. DCW Industries, La Cañada, CA, 3rd edition, 2006.

J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar. Integrating physics-based modeling with machine learning: A survey. *Physics Reports*, 896:1–57, 2020.

R. J. Wilson. *Introduction to Graph Theory*. Addison Wesley, 4th edition, 1996. ISBN 9780582249936.

H. Xiao, J.L. Wu, J.X. Wang, R. Sun, and C. J. Roy. Quantifying and reducing model-form uncertainties in reynolds-averaged navier-stokes simulations: A data-driven, physics-informed bayesian approach. *Journal of Computational Physics*, 324:115–136, 2016.

Y. Xie, E. Franz, M. Chu, and N. Thuerey. tempogan: a temporally coherent, volumetric gan for super-resolution fluid flow. *ACM Transactions on Graphics (TOG)*, 37(4):95, 2018.

J. Yang, T. Dzanic, B. Petersen, J. Kudo, K. Mittal, V. Tomov, J.S. Camier, T. Zhao, H. Zha, T. Kolev, R. Anderson, and D. Faissol. Reinforcement learning for adaptive mesh refinement. In *International Conference on Artificial Intelligence and Statistics*, pages 5997–6014. PMLR, 2023.

Y. Zhao, H. D. Akolekar, J. Weatheritt, V. Michelassi, and R. D. Sandberg. Rans turbulence model development using CFD-driven machine learning. *Journal of Computational Physics*, 411:109413, 2020.