Forschungsbericht 2025-19

Data-Driven Corrections of Low-Fidelity Computational Fluid Dynamics Simulations

Anna Marie Kiener

Deutsches Zentrum für Luft- und Raumfahrt
Institut für Aerodynamik und Strömungstechnik
Standort Braunschweig

Forschungsbericht 2025-19

Data-Driven Corrections of Low-Fidelity Computational Fluid Dynamics Simulations

Anna Marie Kiener

Deutsches Zentrum für Luft- und Raumfahrt
Institut für Aerodynamik und
Strömungstechnik
Standort Braunschweig

147 Seiten
79 Bilder
22 Tabellen
184 Literaturstellen

Herausgeber:

Deutsches Zentrum
für Luft- und Raumfahrt e. V.
Wissenschaftliche
InformationLinder Höhe
D-51147 Köln

ISSN 1434-8454
ISRN DLR-FB-2025-19
Erscheinungsjahr 2025
DOI: 10.57676/53kh-2t16

Datengetrieben, maschinelles Lernen, Strömungssimulationen, diskontinuierliche Galerkin Methode, finite Volumen, Diskretisierung, CFD

Anna Marie KIENER
DLR, Institut für Aerodynamik und Strömungstechnik, Standort Braunschweig

Datengetriebene Korrektur von Strömungssimulationen geringer Genauigkeit
Dissertation, Technische Universität Braunschweig

Computational Fluid Dynamics (CFD) ermöglicht die Simulation komplexer Strömungen und ist in der Luftindustrie weit verbreitet. Trotz ihrer weiten Verbreitung sind CFD Simulationen in großem Maßstab für komplette Flugzeugkonfigurationen nach wie vor eine rechenintensive Aufgabe. Im Allgemeinen erfordern genaue Simulationen eine hohe Auflösung in Form von Freiheitsgraden, um die Strömung in relevanten Bereichen aufzulösen, was mit hohen Rechenkosten verbunden ist. Dies verhindert parametrische Szenarien, bei denen es von Interesse ist, Simulationen für verschiedene Strömungsbedingungen durchzuführen. Traditionell wird versucht, Algorithmen zu beschleunigen, um akkurate Simulationen effizienter zu berechnen. Die vorliegende Arbeit verfolgt den entgegengesetzten Ansatz: die Verbesserung der Genauigkeit von rechnerisch weniger aufwendigen aber dafür ungenauen Simulationen.

Das vorgeschlagene Korrekturverfahren ist datengetrieben, wobei Machine Learning Modelle, die mit genauen Daten trainiert wurden, als Funktionsapproximatoren dienen. Für die Korrektur von stationären Simulationen verfolgt der Ansatz drei Schritte: Zunächst werden genaue und ungenaue Simulationen berechnet, um einen Datensatz zu erzeugen. Anschliessend zielt das Training darauf ab, eine Beziehung zwischen den aus dem Datensatz extrahierten Features und dem Korrekturterm herzustellen. Schließlich wird die Vorhersage des Modells verwendet, um ungenaue Simulationen zu korrigieren. Dieser auf Supervised Learning basierende Ansatz wird auf instationäre Simulationen ausgeweitet und mit einer Methode des Reinforcement Learning verglichen.

Bei stationären Simulationen verbessern die Korrekturen das Strömungsfeld sowie die Oberflächen- und Integralgrößen wie Druck- und Auftriebsbeiwert. Dies wird für turbulente Strömungen auf einem 2D-Profil und einem 3D-Flügel, sowie für laminare Strömungen um einen 3D-Flügel demonstriert. Die Projektion, die notwendig ist um Daten zwischen verschiedenen Diskretisierungen zu übertragen, stellt eine Obergrenze für die erreichbare Genauigkeit dar. Darüber hinaus weisen die trainierten Modelle Generalisierungsgrenzen sowie eine geringere Genauigkeit in der Nähe von Diskontinuitäten auf.

Schließlich wird die vorgeschlagene Methode auf instationäre Probleme angewandt, einschließlich eines linearen Transportproblems und der Konvektion eines isentropen 2D-Wirbels. Reinforcement Learning ist hierbei die teuerste Methode und erreicht gleichzeitig die geringste Genauigkeit. Im Gegensatz dazu erweist sich die einfachste Methode, ein Supervised Learning Ansatz mit entkoppelter Korrektur, innerhalb des gewählten Designraums und Trainingszeitfensters als am zuverlässigsten.

Diese Arbeit zeigt das Potenzial datengetriebener Methoden zur Verbesserung der Genauigkeit von CFD-Simulationen mit geringer Genauigkeit auf. Der vorgeschlagene Ansatz dient als effektive Technik der Ersatzmodellierung für die schnelle Vorhersage in parametrisierten Szenarien.

Data-driven, correction, machine learning, low-fidelity, CFD, discontinuous Galerkin, finite volume, discretization

(Published in English language)

Anna Marie KIENER
German Aerospace Center (DLR), Institute of Aerodynamics and Flow Technology, Braunschweig

Data-Driven Corrections of Low-Fidelity Computational Fluid Dynamics Simulations
Doctoral Thesis, Technical University Braunschweig

Computational Fluid Dynamics (CFD) enables the simulation of complex fluid flows and is widely used within the aerospace industry. Despite its widespread adoption, high-fidelity CFD simulations remain a computationally demanding task for full-aircraft configurations at scale. In general, high-fidelity simulations require high resolution in terms of degrees of freedom to resolve the flow in areas of interest, which is associated with high computational costs. This prohibits many-query scenarios, in which it is of interest to conduct simulations parametrically across various flow conditions.

Traditionally, efforts in CFD focus on improving algorithms to obtain high-fidelity simulations more efficiently. This work takes the opposite approach: improving the accuracy of computationally less expensive low-fidelity simulations, including coarse grid finite volume and low-order discontinuous Galerkin discretizations.

The proposed correction framework is data-driven, where machine learning models, trained on high-fidelity data, serve as function approximators. For the correction of steady simulations, the approach follows three steps: first, a set of high- and low-fidelity simulations is computed to generate a data set. Secondly, the model training aims to find a relationship between the input features and the correction term, which are extracted from the data set. Finally, the model's prediction is used to enhance the accuracy of low-fidelity simulations outside of the training data set. For unsteady simulations, a reinforcement learning approach is proposed and compared to corrections based on supervised learning.

For steady simulations, the corrections improve the flow field, as well as surface and integral quantities such as pressure and lift coefficient. This is demonstrated for turbulent flows using the Reynolds Averaged Navier-Stokes equations with a one-equation turbulence model on a 2D airfoil and a 3D wing, as well as for laminar flow around a 3D wing. However, the need to project high-fidelity data onto low-fidelity discretizations ultimately poses an upper bound to the achievable accuracy, specifically for certain variables of interest such as the drag coefficient. Additionally, the trained models exhibit generalization limits as well as reduced accuracy near discontinuities. Finally, the proposed method is extended to unsteady problems, including a linear transport problem and the convection of a 2D isentropic vortex described by the 1D advection and the Euler equations, respectively. The reinforcement learning approach is the most expensive method, while also achieving the lowest accuracy, accumulating errors due to its autoregressive nature. In contrast, the simplest method, a supervised learning approach with post-processing correction, proves to be most reliable within the chosen design space and training time window.

This thesis highlights the potential of data-driven methods for enhancing the accuracy of low-fidelity CFD simulations. The proposed approaches serve as an effective surrogate modeling technique for rapidly predicting flow field and pressure related variables in parametrized many-query scenarios.

TU Braunschweig – Niedersächsisches Forschungszentrum für Luftfahrt

Berichte aus der Luft- und Raumfahrttechnik

Forschungsbericht 2025-12
DLR-FB-2025-19

Data-Driven Corrections of Low-Fidelity Computational Fluid Dynamics Simulations

Anna Marie Kiener

Deutsches Zentrum für Luft- und Raumfahrt
Institut für Aerodynamik und Strömungstechnik
Braunschweig

Diese Arbeit erscheint gleichzeitig als von der Fakultät für Maschinenbau der Technischen Universität Carolo-Wilhelmina zu Braunschweig zur Erlangung des akademischen Grades eines Doktor-Ingenieurs genehmigte Dissertation.

Data-Driven Corrections of Low-Fidelity Computational Fluid Dynamics Simulations

Von der Fakultät für Maschinenbau
der Technischen Universität Braunschweig
zur Erlangung der Würde

einer Doktor-Ingenieurin (Dr.-Ing.)

genehmigte Dissertation

von:	Anna Marie Kiener, MSc
geboren in:	Luzern, Schweiz
eingereicht am:	13. März 2025
mündliche Prüfung am:	09. Mai 2025
Vorsitz:	Prof. Dr.-Ing. Ulrich Römer
Gutachter:	Prof. Dr. Stefan Görtz
Gutachterin:	Jun.-Prof. Dr.-Ing. Federica Ferraro

2025

Abstract

Computational Fluid Dynamics (CFD) enables the simulation of complex fluid flows and is widely used within the aerospace industry. Despite its widespread adoption, high-fidelity CFD simulations remain a computationally demanding task for full-aircraft configurations at scale. In general, high-fidelity simulations require high resolution in terms of degrees of freedom to resolve the flow in areas of interest, which is associated with high computational costs. This prohibits many-query scenarios, in which it is of interest to conduct simulations parametrically across various flow conditions. Traditionally, efforts in CFD focus on improving algorithms to obtain high-fidelity simulations more efficiently. This work takes the opposite approach: improving the accuracy of computationally less expensive low-fidelity simulations, including coarse grid finite volume and low-order discontinuous Galerkin discretizations.

The proposed correction framework is data-driven, where machine learning models, trained on high-fidelity data, serve as function approximators. For the correction of steady simulations, the approach follows three steps: first, a set of high- and low-fidelity simulations is computed to generate a data set. Secondly, the model training aims to find a relationship between the input features and the correction term, which are extracted from the data set. Finally, the model's prediction is used to enhance the accuracy of low-fidelity simulations outside of the training data set. For unsteady simulations, a reinforcement learning approach is proposed and compared to corrections based on supervised learning.

For steady simulations, the corrections improve the flow field, as well as surface and integral quantities such as pressure and lift coefficient. This is demonstrated for turbulent flows using the Reynolds Averaged Navier-Stokes equations with a one-equation turbulence model on a 2D airfoil and a 3D wing, as well as for laminar flow around a 3D wing. However, the need to project high-fidelity data onto low-fidelity discretizations ultimately poses an upper bound to the achievable accuracy, specifically for certain variables of interest such as the drag coefficient. Additionally, the trained models exhibit generalization limits as well as reduced accuracy near discontinuities. Finally, the proposed method is extended to unsteady problems, including a linear transport problem and the convection of a 2D isentropic vortex described by the 1D advection and the Euler equations, respectively. The reinforcement learning approach is the most expensive method, while also achieving the lowest accuracy, accumulating errors due to its autoregressive nature. In contrast, the simplest method, a supervised learning approach with post-processing correction, proves to be most reliable within the chosen design space and training time window.

This thesis highlights the potential of data-driven methods for enhancing the accuracy of low-fidelity CFD simulations. The proposed approaches serve as an effective surrogate modeling technique for rapidly predicting flow field and pressure related variables in parametrized many-query scenarios.

Zusammenfassung

Computational Fluid Dynamics (CFD) ermöglicht die Simulation komplexer Strömungen und ist in der Luftindustrie weit verbreitet. Trotz ihrer weiten Verbreitung sind CFD Simulationen in großem Maßstab für komplette Flugzeugkonfigurationen nach wie vor eine rechenintensive Aufgabe. Im Allgemeinen erfordern genaue Simulationen eine hohe Auflösung in Form von Freiheitsgraden, um die Strömung in relevanten Bereichen aufzulösen, was mit hohen Rechenkosten verbunden ist. Dies verhindert parametrische Szenarien, bei denen es von Interesse ist, Simulationen für verschiedene Strömungsbedingungen durchzuführen. Traditionell wird versucht, Algorithmen zu beschleunigen, um akkurate Simulationen effizienter zu berechnen. Die vorliegende Arbeit verfolgt den entgegengesetzten Ansatz: die Verbesserung der Genauigkeit von rechnerisch weniger aufwendigen aber dafür ungenauen Simulationen.

Das vorgeschlagene Korrekturverfahren ist datengetrieben, wobei Machine Learning Modelle, die mit genauen Daten trainiert wurden, als Funktionsapproximatoren dienen. Für die Korrektur von stationären Simulationen verfolgt der Ansatz drei Schritte: Zunächst werden genaue und ungenaue Simulationen berechnet, um einen Datensatz zu erzeugen. Anschliessend zielt das Training darauf ab, eine Beziehung zwischen den aus dem Datensatz extrahierten Features und dem Korrekturterm herzustellen. Schließlich wird die Vorhersage des Modells verwendet, um ungenaue Simulationen zu korrigieren. Dieser auf Supervised Learning basierende Ansatz wird auf instationäre Simulationen ausgeweitet und mit einer Methode des Reinforcement Learning verglichen.

Bei stationären Simulationen verbessern die Korrekturen das Strömungsfeld sowie die Oberflächen- und Integralgrößen wie Druck- und Auftriebsbeiwert. Dies wird für turbulente Strömungen auf einem 2D-Profil und einem 3D-Flügel, sowie für laminare Strömungen um einen 3D-Flügel demonstriert. Die Projektion, die notwendig ist um Daten zwischen verschiedenen Diskretisierungen zu übertragen, stellt eine Obergrenze für die erreichbare Genauigkeit dar. Darüber hinaus weisen die trainierten Modelle Generalisierungsgrenzen sowie eine geringere Genauigkeit in der Nähe von Diskontinuitäten auf.

Schließlich wird die vorgeschlagene Methode auf instationäre Probleme angewandt, einschließlich eines linearen Transportproblems und der Konvektion eines isentropen 2D-Wirbels. Reinforcement Learning ist hierbei die teuerste Methode und erreicht gleichzeitig die geringste Genauigkeit. Im Gegensatz dazu erweist sich die einfachste Methode, ein Supervised Learning Ansatz mit entkoppelter Korrektur, innerhalb des gewählten Designraums und Trainingszeitfensters als am zuverlässigsten.

Diese Arbeit zeigt das Potenzial datengetriebener Methoden zur Verbesserung der Genauigkeit von CFD-Simulationen mit geringer Genauigkeit auf. Der vorgeschlagene Ansatz dient als effektive Technik der Ersatzmodellierung für die schnelle Vorhersage in parametrisierten Szenarien.

Acknowledgements

I would like to express my gratitude to everyone who accompanied and supported me over the past four years.

Thanks to my committee. First, my doctoral supervisor, Prof. Dr. Stefan Görtz, whose support made it possible for me to pursue this PhD. I also thank Jun.-Prof. Dr.-Ing. Federica Ferraro for co-examining my dissertation.

Special gratitude goes to Dr. Philipp Bekemeyer and Prof. Dr. Stefan Langer. Professionally, I learned so much from you, both from an engineering and mathematical perspective. But beyond that, you've given me invaluable personal guidance. *"We make a living by what we get, but we make a life by what we give."* - and the two of you gave me a lot.

I am also grateful to the Surrogates and Uncertainty Management team for the friendly atmosphere and insightful discussions. A special mention to Andrea and Simon: the former haunted me even after leaving the DLR, pushing me to complete my writing, while the latter was by my side from start to finish, a steady companion throughout this journey. Additionally, I am very happy that Sihyeong decided to join our team and became a great office mate during my last year.

I deeply appreciate the welcoming environment in the Institute of Aerodynamics and Flow Technology, especially in the $C^2A^2S^2E$ department I was a part of. A heartfelt thanks goes to Axel Schwöppe for his support of young researchers, and for the encouragement he gives us.

My understanding of discontinuous Galerkin methods would not be the same without Malte, Miquel, and Thomas. During my stay in Paris, the colleagues at ONERA, especially Florent and Jean-Baptiste, provided invaluable guidance.

Many thanks go to the Covfefe Kids: Anjali, Fabi, Jesús, Malte, Miquel, as well as Wojciech and the pebbles. A PhD journey comes with its ups and downs, but no matter how challenging, coming to the office was always a pleasure because of our conversations.

I greatly appreciate the time I spent with friends outside of work - especially over good food in skillets, at the farmers market, during move nights, at hikes, or at Greifhaus. One friendship I wish to highlight is the one I found with Miquel.

A heartfelt thank you to Anjali, who became a wonderful friend along the way. I have no doubt our friendship will last well beyond our time at the DLR. May our giggles forever echo in Philipp's ears.

I want to thank my friends and family back in Switzerland: my mother, who always believes in me, my sister Marnie, to whom I've always looked up to, Livio, who is like a brother to me, and Manuela, whose voice messages brighten my days.

Finally, my deepest gratitude goes to my partner, Serjosha. We pursued our academic journeys side by side, you in Zurich and I in Braunschweig, supporting each other every step of the way. I am grateful for this shared experience, but even more so, I look forward to our paths converging, spending time together on the things that truly matter in life.

List of Publications and Presentations

The work presented in this thesis has led to several publications and presentations, with details provided below.

Journal Papers

- A. Kiener, S. Langer, P. Bekemeyer, Data-driven correction of coarse grid CFD simulations, Computers & Fluids, Volume 264, 2023, 105971, ISSN 0045-7930, DOI: https://doi.org/10.1016/j.compfluid.2023.105971
- A. Kiener, P. Bekemeyer, Correcting Unsteady Low-Order Discontinuous Galerkin Simulations, AIAA Journal, 2025, under review since 07.02.2025
- A. Kiener, S. Langer, P. Bekemeyer, Data-Driven Correction of Low-Order Discontinuous Galerkin Simulations, Computers & Fluids, 2025, to be submitted

Conference Proceedings

- A. Kiener, P. Bekemeyer, S. Langer, Correcting the Discretization Error of Coarse Grid CFD Simulations with Machine Learning, ECCOMAS Congress 2022, Oslo, DOI: 10.23967/eccomas.2022.072
- A. Kiener, P. Bekemeyer, S. Langer, Towards Fast Aerodynamic Simulations with Machine Learning Corrections for Discretization Errors, DLRK Hamburg, 2024, DOI: 10.25967/630064
- A. Kiener, P. Bekemeyer, Correcting Unsteady Low Order Discontinuous Galerkin Simulations, AIAA SciTech 2025 Forum, Orlando, 2025, DOI: https://doi.org/10.2514/6.2025-2046, AIAA paper 2025-2046
- P. Bekemeyer, A. Bertram, D. A. Hines Chaves, M. Dias Ribeiro, A. Garbo, A. Kiener, C. Sabater, M. Stradtner, S. Wassing, M. Widhalm, S. Görtz, F. Jaeckel, R. Hoppe, N. Hoffmann, Data-Driven Aerodynamic Modeling Using the DLR SMARTy Toolbox, AIAA Aviation 2022 Forum, Chicago, 2022, DOI: 10.2514/6.2022-3899, AIAA paper 2022-3899
- C. Fallet, A. Garbo, A. Kiener, P. Bekemeyer, Reduced Order Model for Steady Aerodynamics Applications Based on Navier-Stokes Residual Vector Minimization, AIAA Aviation 2023 Forum, San Diego, 2023, DOI: 10.2514/6.2023-3715, AIAA paper 2023-3715

Presentations

- A. Kiener, Use of Machine Learning Methods during Post-Processing to Reduce the Discretization Error of Coarse Grid CFD Results, Advances in Artificial Intelligence for Aerospace Engineering 2nd workshop, 2021, Online
- A. Kiener, P. Bekemeyer, Correcting the Discretization Error of Coarse Grid CFD Simulations with Machine Learning, 22nd ODAS: ONERA - DLR Aerospace Symposium, 2022, Hamburg,
  Awarded with *"The best paper prepared and presented by young scientists"*
- A. Kiener, Solver Consistent Data-Driven Correction of Coarse Grid CFD Simulations, Advances in Artificial Intelligence for Aerospace Engineering 3rd workshop, 2023, Paris
- A. Kiener, F. Renac, Learning Corrections of Discontinuous Galerkin Schemes, Machine Learning for Fluid Dynamics Workshop, ERCOFTAC, 2024, Paris
- A. Kiener, F. Renac, Learning Corrections of Low-Order Discontinuous Galerkin Schemes, Advances in Artificial Intelligence for Aerospace Engineering 4th workshop, 2024, Braunschweig
- A. Kiener, Data-Driven Corrections of CFD Simulations, AI4Science Talks - invited talk by University of Stuttgart and NEC Labs Europe, 2024, Online

Contents

List of Figures

List of Tables

Nomenclature

AMR	Adaptive Mesh Refinement
AD	Automatic Differentiation
CFD	Computational Fluid Dynamics
CNN	Convolutional Neural Network
DEM	Discrete Element Method
DES	Detached Eddy Simulation
DG	Discontinuous Galerkin
DNS	Direct Numerical Simulation
DRL	Deep Reinforcement Learning
FOM	Full Order Model
FV	Finite Volume
GNN	Graph Neural Network
HO	High-order
HPO	Hyperparameter Optimization
HVAC	Heating, Ventilation, and Air Conditioning
i.i.d.	independent and identical distributed
ICAO	International Civil Aviation Organisation
KL	Kullback-Leibler
LES	Large Eddy Simulation
LO	Low-order
MAE	Mean Absolute Error
MDP	Markov Decision Process
MG	Multigrid
ML	Machine Learning
MSE	Mean Squared Error
NN	Neural Network
o.o.d.	out-of-distribution
PDE	Partial Differential Equation
POD	Proper Orthogonal Decomposition
POMDP	Partial Observable Markov Decision Process
PPO	Proximal Policy Algorithm
RANS	Reynolds-Averaged Navier-Stokes
RF	Random Forest
RK	Runge-Kutta
RL	Reinforcement Learning
ROM	Reduced Order Model
SA	Spalart-Allmaras
TPE	Tree Parzen Estimator
TRPO	Trust Region Policy Optimization

Symbols

Greek Symbols

α	Angle of attack
β	Vortex strength
δ	Action
ε	Error
γ	Discount factor
μ	Mean
μ_{dyn}	Dynamic viscosity
μ_l	Laminar viscosity
μ_t	Eddy viscosity
ν	Molecular viscosity
$\tilde{\nu}$	Turbulent variable
π	Policy
ψ	Critic parameters
ϕ	Polynomial basis function
ρ	Density
τ	Viscous stress tensor
τ_w	Wall shear stress
θ	Machine learning parameters
σ	Standard deviation
Θ	Random vector
Ω	Bounded domain
Ω_h	Decomposition of Ω

Roman Symbols

a	Polynomial coefficient
$\boldsymbol{b}$	Bias vector
c	Constant velocity coefficient
d	Problem dimension
e	Edge
g	Activation function
h	Discretization
$\boldsymbol{h}_k$	Feature vector representation of vertex k
k	Element or control volume
k_T	Thermal conductivity
n	Normal
p	Polynomial degree
p_{tr}	Transition probability
$\boldsymbol{q}$	Conductive heat flux

r	Reward
s	State
t	Time
u	Conserved quantity
$\bar{u}$	Mean of u
u'	Fluctuating component of u
$\tilde{u}$	Approximation of u
$\hat{u}$	Machine learning corrected u
v	Vertex
x	Coordinate
y	Machine learning outputs
$\hat{y}$	Machine learning predicted outputs
$\boldsymbol{\eta}$	Machine learning feature input vector
A_π	Advantage function under policy π
C	Sutherland's constant
C_f	Skin friction coefficient
C_L	Lift coefficient
C_P	Pressure coefficient
D	Number of decision trees
D_t	Turbulent destruction term
E	Total energy
$\mathcal{E}$	Edges
F_c	Vector of convective fluxes
F_v	Vector of viscous fluxes
G	Return
$\mathcal{G}$	Graph
H	Horizon
H_{tot}	Total enthalpy
I	Number of elements
I_f^c	Mapping from fine to coarse grid
J	Number of labeled input and output pairs
$\mathcal{L}$	Loss function
L	Number of neural network layers
L_v	Vortex length
L_{ref}	Reference length
M	Mach number
M_x	Momentum in x direction
N	Number of degrees of freedom
$\mathcal{N}$	Set of neighborhood vertices of vertex
P	Pressure
P_r	Probability ratio
P_t	Turbulent production term
Q_π	Action-value function or Q-function under policy π
R	Ideal gas constant

R^2	Coefficient of Determination
Re	Reynolds number
S	Surface
T	Temperature
U	Velocity
V_π	Value function under policy π
V	Vertices
V_h^p	Space of polynomials of total degree p
$\boldsymbol{W}$	Weight matrix

Chapter 1

Introduction

The ultimate goal is to obtain the desired accuracy with the least effort, or the maximum accuracy with the available resources.

Ferziger, Peric, Street [52]

1.1 Motivation: The Relationship Between Efficiency and Accuracy

Technical advancements have led to cheaper and more accessible air mobility. This has facilitated globalization, driving many of today's achievements. However, the increased use of air transport comes with a growing need for infrastructure and contributes to noise and pollution. Policymakers face the challenge of balancing the growing demand for mobility with the urgent need to combat climate change. One way of reducing aircraft emissions and thus the contribution to climate change is by technical means [64], i.e. the pursuit of green aviation. Previously, one of the primary objective of aerospace engineers was to improve cost efficiency. However, in response to recent global developments, the focus has fundamentally shifted towards sustainability as a critical priority.

In this context, Computational Fluid Dynamics (CFD) plays a crucial role in the development of new aircraft. CFD entails the computational simulation of fluid flow behavior and has become an essential element in many industrial fields. For instance, in the automotive industry, CFD is used for the design of combustion engines and the analysis of aerodynamic forces. In the Heating, Ventilation and

Air Conditioning (HVAC) industry, CFD is employed to analyze indoor airflow, heat transfer and temperature distribution. These are just a few examples to show the broad scope of CFD applications, which provide insights into complex fluid flows. The initial promise of CFD in the aerospace industry was to substitute expensive tests, thereby saving time and resources in designing new and improved aircraft, long before a physical model or prototype is built. As of today, while CFD has matured to a state in which it reduces the need for certain physical tests, it has not replaced expensive wind tunnel experiments or flight tests, but rather complements these methods during all stages of design. This can be attributed to the fact that full-scale simulations of aircraft still remain a computationally demanding task. Reducing computational cost of CFD simulations remains an active area of research, and this thesis aims to contribute to these efforts.

The focus of this thesis is to efficiently approximate solutions of fluid flow problems described by boundary value problems. In general, scientist and engineers mathematically describe physical problems, be it in the domain of fluid dynamics or other, using Partial Differential Equations (PDE), where the unknown function depends on both space and time. Consider as a simple example the heat equation, given as

$$\frac{\partial u}{\partial t} = c\frac{\partial^2 u}{\partial x^2}, \tag{1.1}$$

with x and t the independent variables representing the space and time dimensions, u being the variable depending on x and t, and c a real positive constant. To arrive at the boundary value problem for which one desires a solution, auxiliary conditions at the boundary need to be stated. While analytical solutions are desirable, they are usually derived only for very simple problems. In most practical applications, especially those involving complex geometries or non-linear terms in the PDEs, which is often the case for fluid problems, finding an analytical solution is generally difficult. As a result, numerical methods are commonly used to find an approximation of the analytical solution. Employing the method of lines, all but one derivative, typically the spatial ones, are discretized, using for example an algebraic approximation such as finite differences. With this, the original PDE is converted into a system of time dependent Ordinary Differential Equations (ODE), which can be integrated with a suitable time stepping scheme. As fluid problems are of interest here, for which the governing equations are of non-linear nature, the considered spatial discretization schemes are the Finite Volume (FV) [19] and the Discontinuous Galerkin (DG) [36] methods. The FV discretization has become the dominant one throughout several fluid related applications, whereas the DG discretization, being a higher order method receiving currently more attention from academia than from industry, is often dismissed for being too expensive [20, 168].

Although the increased use of CFD simulations has lead to a variety of such discretization schemes, all of them share the same trade-off: accurate solutions are accompanied by increased computational cost. This ultimately is a bottleneck for performing a large number of high-fidelity simulations. Increasing the accuracy of CFD simulations necessitates in general more degrees of freedom N. For both

FV and DG discretization, the physical domain of interest is first divided into cells, collectively forming the mesh or grid. In the case of FV, the number of elements or vertices corresponds to the number of degrees of freedom N. To accurately capture phenomena of interest, such as small-scale turbulence near an airfoil, the mesh must be refined in these regions. In contrast, less relevant areas, such as the far field, require less resolution. As the DG discretization describes the solution in one element as a piecewise polynomial approximation, it allows to control the number of degrees of freedom N not only by refining the grid, but also by increasing the polynomial degree. While increasing the polynomial degree results in a rapid growth of degrees of freedom, it is notable that choosing the lowest possible polynomial degree results in a first order FV scheme.

For the scope of this thesis, it is necessary to clarify and define certain terms, including *computational cost*, *accuracy*, as well as *low-* and *high-fidelity*. Firstly, computational cost can be regarded in terms of complexity, putting the number of degrees of freedom N into relation with the time needed to obtain a numerical solution, as given in Figure 1.1. Obviously, it would be desirable that the solution algorithms scale linearly with the number of degrees of freedom N, such that $\mathcal{O}(N)$ is achieved. In reality, obtaining numerical solutions of the Reynolds-Averaged Navier-Stokes equations, which are the central governing equations within this work, has the computational complexity $\mathcal{O}(N^m)$, with $m \geq 2$, while pre-conditioning methods can lessen this complexity to a certain extent. Thus, while increasing the number of degrees of freedom on the one hand *might* improve the solution accuracy, the computational cost or time to obtain a solution *must* increase at least quadratically on the other hand.

Then, the question arises: how many degrees of freedom are necessary to achieve sufficient accuracy? Naturally, it is desirable for a numerical solution u_h to approximate the analytical solution u, such that the error

$$\varepsilon(h) = |u - u_h|, \tag{1.2}$$

being a function of the chosen discretization h, vanishes. In this work, an accurate solution either satisfies convergence in terms of degrees of freedom, meaning that the solution does not change significantly with increased N, or approximates a

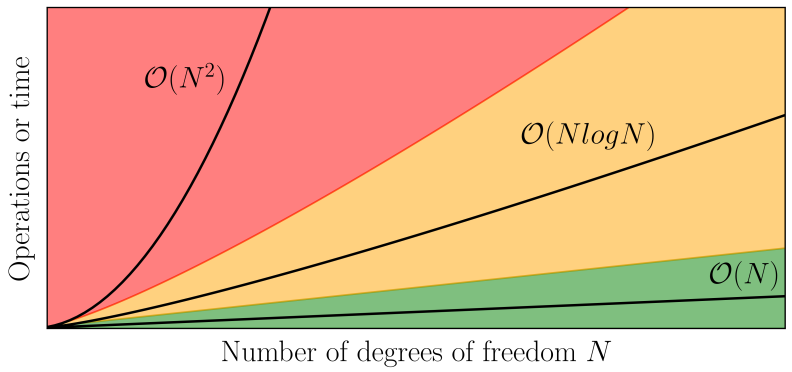


Figure 1.1: Computational cost in terms of complexity

reference value. Thus, the obtained solution is deemed accurate if it is comparable either to the analytical solution if available, or measurement or simulation data from literature. These accurate solutions are labeled as high-fidelity, and are obtained for FV and DG on relatively fine grids or with high polynomial degrees, respectively. Contrarily, low-fidelity solutions are obtained with less degrees of freedom, either on coarser grids or with lower polynomial degrees, with respect to their high-fidelity counterparts. With the established complexity of at least $\mathcal{O}(N^2)$, these low-fidelity solutions are less expensive, but with the prospect of being less accurate.

Two obvious ways to decrease the time needed to obtain high-fidelity solutions is to either improve the algorithms or the respective hardware capabilities. Promising algorithms include multigrid methods [23] and adaptive refinement of the grid or the polynomial degree [16]. Regarding hardware, there is a trend towards higher parallelism [140] and heterogeneous architectures [184], although it can be shown that the parallel efficiency of many solution algorithms scale sub-optimally [104]. Also, a renewed interest in utilizing GPUs for CFD computations can be observed [21, 138, 179].

Another approach to achieve accurate solutions in less time is by enhancing the accuracy of the low-fidelity solution. This is often the starting point for current research aimed at accelerating CFD solutions through data-driven methods. Machine Learning (ML) methods, in particular, have received increased attention across various research fields in recent years. ML, especially deep learning, has achieved notable success in solving problems related to image recognition [129] and natural language processing [128]. Thus, there is considerable interest in achieving similar success for physics-based problems, well outlined in various review papers [27, 46, 163, 58], and this work builds upon those efforts.

To summarize, there exists a relationship between complexity and accuracy in CFD simulations, while the demand for achieving accurate solutions in less time is rising. Nowadays, increasingly complex problems are tackled, including multi-disciplinary simulations. Here, engineers must not only account for aerodynamics, but a multitude of disciplines, such as structural integrity and propulsion. Quick and precise design iterations are essential for exploring various design options and optimizing performance under diverse conditions. This is particularly the case in the context of many-query problems, where numerous simulations in a parametrized setting must be conducted efficiently. The ability to efficiently simulate complex, mutual dependent phenomena is crucial for advancing aerospace engineering and meeting the industry's evolving demands and the goal of green aviation. Thus, to advance simulation capabilities, there is a necessity for "revolutionary algorithmic improvements", as the CFD vision 2030, published by NASA in 2014, states [153].

1.2 State of the Art

This section discusses established methods, such as multigrid methods and adaptive refinement, and closes with an overview of current research trends involving ML methods. This review of the state of the art provides the foundation for this thesis, exploring if and how ML methods can improve the accuracy of low-fidelity CFD simulations.

1.2.1 Classical Algorithms to Accelerate CFD Simulations

As previously described, low-fidelity simulations comprising less degrees of freedom tend to be computational inexpensive while potentially being less accurate. However, certain algorithms exploit these properties to their advantage.

One such technique is the Multigrid (MG) method, which requires to transfer information between different discretizations by using projection and interpolation operators. Using Fourier expansion, it can be shown that the solution error which is reduced during the iterative solving procedure contains high- and low-frequency components, while the latter is the main driver for slow convergence [65, 161]. The fundamental idea of MG is to smoothen the error. By employing a hierarchy of discretizations, such as a sequence of coarse to fine grids, the costly low-frequency error components of the fine grid discretization become high-frequency components on the coarse grid discretization. On the low-fidelity discretization, the error can be more efficiently reduced, promising to accelerate the overall convergence behavior [19].

Another method that leverages the fast solution of low-fidelity discretizations is the adaptive refinement of a mesh or increase of a polynomial degree, known as h- or p-refinement, respectively. Adaptive Mesh Refinement (AMR) has found applications in various areas such as climate modeling and astrophysics [43]. The core idea of AMR is to dynamically refine the discretization locally where needed, while achieving a coarse discretization otherwise, to efficiently arrive at an acceptable accuracy with only as many degrees of freedom as needed. In fluid problems, critical areas in need of refinement are encountered where the solution exhibits steep gradients, such as inside the boundary layer or at discontinuities [16]. The AMR algorithm typically starts with a coarse mesh. After obtaining a solution on this discretization, an error indicator flags cells for refinement. The choice of an indicator is non-trivial, with a wide variety existing in literature, including gradient- or residual-based indicators [90]. Repeating the refinement on the next discretization leads to a sequence of grids. This iterative process of solving, error estimation, and mesh refinement is repeated until a certain stopping criterion is satisfied. The same procedure applies to p-refinement, where the polynomial degree within the flagged elements is increased instead of the element being refined, while a combination of both, known as hp-refinement, also exists.

The discussed methods have demonstrated their effectiveness, showing that the use of low-fidelity discretizations can accelerate the process of obtaining accurate solutions [82, 177]. Nevertheless, Witherden and Jameson [172] argue that CFD progress has plateaued since the early 2000s, with second-order FV schemes currently being the state of the art for solving steady flow problems. They emphasize that the practical use of CFD tools is closely linked to the speed and memory capacity of available computational hardware, and thus CFD benefits directly from advancements in hardware technology. However, trends like heterogeneous computing, higher parallelism, and exploiting GPU architectures come with obstacles, including increased development and maintenance cost, while small engineering companies are constrained not only by the number of software licenses but also by the available hardware. Additionally, the efficiency gains are often diminished by the need of on- and offloading data between different devices and load balancing cost. Most importantly, from an algorithmic perspective, it has been shown that several schemes lose efficiency with increased levels of parallelism [104]. Therefore, with the given increase in problem complexity and demand of simulations, there is a need for innovative algorithmic approaches that can fully and efficiently exploit the potential of current and future hardware trends, highlighting the necessity to improve existing algorithms or even derive new ones. In the light of exploring methods beyond classical algorithms, ML methods have received increased attention across various research fields.

1.2.2 Machine Learning Algorithms to Accelerate CFD Simulations

ML has achieved remarkable success in areas where deep learning models exploit extensive datasets [34]. Examples include object detection in computer vision for autonomous driving [56], generating texts using large language models in natural language processing [66], and discovering new protein structures in biology [80]. Given this progress, there is growing interest in achieving similar achievements for physics-based problems. Trying to leverage synergies between ML techniques and CFD methods, different applications have emerged and are focal point of current research. This is reflected in the number of review articles investigating the application of ML methods for CFD in general [27, 29, 58, 68, 130, 163] or specifically for turbulence modeling [11, 46, 47]. One can categorize ML for physics simulation into *prediction* and *correction* tasks [114]: prediction methods seek to directly forecast the CFD solver's output, bypassing the costly CFD evaluation. Correction methods aim to improve simulation results or certain components within the CFD solver, for example turbulence models. In addition to these two distinctions, the following sections review current research on coupling CFD solvers with ML techniques, a topic that has become central to unsteady simulations.

Predicting CFD Simulations

Surrogate models and Reduced Order Models (ROM) are examples for inexpensive predictions of CFD solutions. During the early stages of design development, rapid evaluations of possible designs allow to iterate through a plethora of geometries and assess them under varying flight conditions. Here, high-fidelity CFD simulations slow down evaluations and thus hold back quick optimization processes. The aim of surrogate models it to predict with sufficient accuracy certain quantities of interest, such as lift, drag or surface coefficient, at a much lower computational cost. As for ROMs, any simplified model could be regarded as a ROM of a higher fidelity model. Thus, the Euler equations could for example be by definition a ROM of the Reynolds-Averaged Navier-Stokes (RANS) equations. Nevertheless, most ROMs in literature rely on data-driven approaches and capture the physical characteristics of the high-fidelity or Full Order Model (FOM) in a reduced or compressed format. This compression is accomplished by selecting an appropriate projection that encapsulates the main properties of the FOM data within a space of lower dimension [117]. Creating the FOM data and fitting the projection to a large dataset are the costly parts of creating a ROM. After this fitting or training, the model is employed for new predictions, which is in turn cheap and allows for fast design iterations. In CFD applications, ROMs are often build on Proper Orthogonal Decomposition (POD) [158, 55, 49]. POD is a dimensionality reduction method, which uses eigenvalue analysis to decompose a vector field, such as the velocity or pressure field, into a set of basis functions. Finally, the solution can be reconstructed from these extracted POD modes.

Recently, deep learning models have gained popularity as surrogates and have shown improved capabilities for certain problems. For instance, Hines et al. [74] compare a POD with an interpolation method, a deep neural network, and a graph neural network, predicting the pressure coefficient based on the Mach number and angle of attack for a wing geometry. It was found that the deep learning models result in higher accuracy for transonic flows. Sabater et al. [144] predict the surface pressure distribution of an airfoil and an aircraft geometry by comparing Gaussian Processes, POD with interpolation, and a deep neural network. Similarly, they concluded that all models achieve high accuracy for subsonic regimes, whereas at transonic design points, where discontinuities such as shocks form, deep learning methods outperform the other approaches.

Correcting CFD Simulations

For aerodynamic applications, the use of previously described surrogates is often limited to the prediction of surface and integral coefficients, while full field simulation data is discarded. Leveraging all available data for ML model corrections allows to address a wider range of problems and ultimately enables the combination of data-driven models with the CFD solver.

One area of correcting CFD simulations is the development of data-driven turbulence models. Resolving all turbulent scales, known as Direct Numerical Simulation (DNS), is currently infeasible for most industrial applications because of high computational cost. Various turbulence models exist for different problems, and the prospects of finding a universal turbulence model seem marginal. For instance, the one-equation Spalart-Allmaras turbulence model is mainly used in aerospace applications [155], while the two-equation $k - \epsilon$ turbulence model is a popular choice for the simulation of indoor airflow [33]. Turbulence models typically contain multiple coefficients that are empirically chosen and carefully calibrated on canonical test cases. In this sense, turbulence models can already be considered somewhat data-driven. However, tuning these models only with canonical cases prohibits them to be accurate for a broad range of complex tasks. By leveraging data-driven methods, such as ML, the aim is to extract new patterns from more diverse datasets, with which existing turbulence models can potentially be augmented. Wang et al. [166] for example train a random forest algorithm to predict the discrepancy in the Reynolds stress between high-fidelity DNS data and the respective low-fidelity RANS simulation. Using mean flow features as inputs, based on previous work by Ling and Templeton [113], improved performance was observed in flows with fully developed turbulence in a square duct with varying Reynolds number and flows with separation over periodic hills with varying geometry. Another approach, commonly called Field Inversion and Machine Learning, is a data-driven paradigm for turbulence model augmentation coined by Parish and Duraisamy [131], and further explored by several research groups [17, 51, 75, 146, 176]. During the field inversion step, a spatial correction field describing the difference between the turbulence model to be corrected and reference data, is inferred, often using an adjoint-driven optimization algorithm. In the second step, a regression model is fitted to predict this correction term, which can then be used to augment the turbulence model equation, for instance by a local factor to the turbulent production term.

Another research area focuses on using ML corrections to accelerate high-fidelity simulations. One such popular method, called super-resolution, is adapted from image processing. Essentially, super-resolution aims to enhance the resolution of under-resolved or blurry images by increasing pixel density to capture more details [132]. This method has been studied in different fields, including medical image diagnosis [6], satellite imaging [136], surveillance [91], and astronomical imaging [87]. Convolutional Neural Networks (CNNs) are commonly used to learn the reconstruction from low- to high-resolution, leveraging their ability to recognize patterns and details in image data [127]. CNNs were first applied to the task of recognizing handwritten digits by LeCun et al. [109]. Since then, they have become a standard tool for image-related tasks because of their efficient handling of high-dimensional data. In CFD, the goal of super-resolution is to reconstruct high-fidelity flow fields from their low-fidelity counterpart. The low-fidelity discretization is analogous to an under-resolved image, with each degree of freedom representing a pixel in the image. For instance, Fukami et al. [57] employ two different models containing convolutional blocks to learn the reconstruction of DNS data for laminar and turbulent flows from the downsampled DNS data. Both models successfully reconstruct the wake of a laminar flow behind a cylinder and improve accuracy for two-dimensional decaying turbulence, judged by the turbulent kinetic energy spectra, velocity, and vorticity fields. Romano and Baysal [142] use a CNN autoencoder to learn the reconstruction from unsteady RANS simulations to Detached-Eddy-Simulations (DES) for different NACA airfoils and varying angles of attack. The model performs well on new angles of attack but its performance significantly decreases on geometries unseen during the training phase. Gao et al. [60] explore the use of CNNs for super-resolution in CFD simulations for cardiovascular systems, reconstructing high-fidelity velocity fields. By introducing a physics-informed loss to minimize PDE residuals, they achieve promising results even with noisy low-resolution data and are able to avoid the need of high-resolution samples for the supervised training. However, several limitations are reported, including the restriction of CNNs to simple geometries with image-like uniform data structures. Morimoto et al. [122] report similar conclusions, highlighting that traditional CNNs are not easily applied to practical CFD problems, since unstructured meshes are commonly encountered in CFD to accurately represent complex geometries. One of the solutions proposed by the authors are Graph Neural Networks (GNN).

Indeed, GNNs have recently gained significant attention in various fields, including physics, where they are used to model particle interaction such as collisions [79], and chemistry, where GNNs predict chemical reactions based on molecular structures [45]. As the name suggests, GNNs operate on graph-structured data, such as social networks, where graph vertices represent people and graph edges their connections. Graphs can also contain vertex and edge features - in the example of a social network, vertex features might include age, while edge features describe the relationship between people, such as colleagues or family members. Similarly, a mesh with a CFD solution can be described as a graph: mesh nodes or cell centres are graph vertices, and their connections graph edges. Field variables like velocity or pressure act as vertex features, while edge features can include distance or angle between the vertices. Similar to CNNs, GNNs convolve information from local parts of the graph by efficiently including neighborhood features through message-passing, thus leveraging the graph-structured data to make predictions. He et al. [72] utilize a graph convolution attention network to predict the complete flow field and the forces on a cylindrical body based on incomplete and unstructured data, highlighting improved results compared to traditional CNNs across varying Reynolds numbers. On body-fitted triangular grids, Chen et al. [32] trained a graph-based surrogate model on incompressible laminar flow around 2'000 different 2D shapes, testing the prediction of velocity and pressure field data as well as the resulting forces on a NACA0012 airfoil. They concluded that compared to classical CNN models, the implementation and training of a GNN is more complex and time consuming, nevertheless it offers higher accuracy and is ultimately a better choice for complex geometries. Other sources include Ogoke et al. [126], predicting the drag force induced by laminar flow around varying airfoils based on sparse velocity measurements, and Wang et al. [167], proposing a framework to identify vortices on unstructured grids, showcasing the versatility of graph-based frameworks. Furthermore, Hines et al. [74] use GNNs to predict the surface pressures of both 2D and 3D geometries, showcasing the scalability of the model. In summary, the relevance of GNNs for CFD is twofold: firstly, they are applicable to graphs of arbitrary structure, which is the case for unstructured grids, since GNNs are permutational invariant to the number of input nodes and edges, something with which traditional neural networks or CNN architectures struggle with. Secondly, they incorporate the topology of the graph, such that their predictions consider local neighborhood information and thus the relation between data points, similarly to stencils used in CFD computations.

Yet another research field aiming to correct CFD simulations involves the use of ML methods to improve the accuracy of low-fidelity simulations. Here, the highly inverse problem of projecting a low-fidelity solution to a high-fidelity discretization, namely super-resolution, is avoided. This, since super-resolved results often provide qualitatively good outcomes suitable for image-related tasks, but lack the quantitative accuracy required for CFD applications. For example, Fukami et al. [59] state in their survey on super-resolution for fluid flows that it cannot be expected to recover DNS data from LES input. Similarly, Shin et al. [151] conclude that super-resolution may introduce new or amplify existing features, making the results potentially misleading, especially in safety critical domains. Thus, the key idea of the following work is to preserve the low simulation cost while injecting high-fidelity information into inaccurate solutions through a data-driven model. Davydzenka and Tahmasebi [40] investigate this concept for fluid-particle interactions, for which CFD and the Discrete Element Method (DEM) are coupled. The trained neural network model predicts for coarse grid simulations the corrected drag forces needed to compute the particle-fluid momentum exchange term. They reported significantly improved results across various metrics, including the run-off distance of particles on coarse meshes with twice and four times increased cell size compared to the ground truth model, while results diminish with an eight-fold coarsened grid, as this is a discretization too coarse to capture any relevant physical phenomena. Hanna et al. [69] propose to directly correct the error induced by the coarse grid, i.e. the discretization error, which can be quantified by mapping the high-fidelity solution onto the coarse grid. Using local flow features from the coarse grid solution, such as a local Reynolds number and the first and second gradients of the flow field variable to be corrected, two ML models are trained to predict the grid-induced error locally. In their case study, a random forest and a neural network are tested to correct the velocity field of the turbulent flow in a three dimensional lid-driven cavity. Varying the Reynolds number, grid spacing, and aspect ratio, they cover cases to test the models' inter- and extrapolation capabilities. Lee and Cant [110] adopt this approach with a random forest model for a turbulent flow around a bluff-body in an enclosed duct. By incorporating additional features related to Reynolds stress and regions characterized by steep pressure and shear gradients, they increase the prediction accuracy of their model. Similarly, Cantarero-Rivera et al. [30] use a neural network to learn and correct coarse grid-induced errors in CFD simulations of mixing flows inside spinner flask bioreactors. They aim to correct the velocity field and the Kolmogorov length across varying fluid viscosity, mixing speeds, and impeller geometry conditions. Despite some limitations in generalization capabilities, all three studies report reduced errors on coarse grids while increasing computational efficiency. This method can be regarded as purely data-driven and decoupled from the CFD solver, as the correction itself is applied in a post-processing fashion after the simulations have been conducted. The disadvantage of decoupled data-driven corrections is the disregard of conservation laws, potentially leading to nonphysical results. To address this issue, further research focuses on integrating both ML training and ML corrections within the CFD framework. These approaches aim to leverage available information from the solver, with unsteady problems often at their core.

Coupling Machine Learning and CFD Solver for Unsteady Problems

Using methods to increase accuracy while maintaining the efficiency of low-fidelity discretizations is of special interest for time variant problems, since unsteady flow simulations are particularly expensive. Although accuracy may suffer, low-fidelity discretizations allow to solve for less degrees of freedom with a greater time step, ultimately decreasing the computational cost. De Lara and Ferrer [42] make use of this fact, employing a neural network to predict corrections of low-order DG simulations. Their method consists of three steps: firstly, data is being collected by advancing both low and high-order simulations up to a certain time. During this, a corrective forcing term is computed representing the difference between both trajectories. Secondly, a neural network is trained to predict this correction. Finally, only the low-order simulation is evolved and the trained network predicts the correction used to approximate the high-order trajectory. They report different sources of error, and conclude that the most significant is the error of the ML model prediction itself, since it accumulates and thus grows over time due to its autoregressive nature. Nevertheless, based on the promising results for smooth solutions of the 1D Burgers' equation, the proposed approach is tested on the 3D Taylor Green Vortex problem [118]. They report an increased acceleration of 4-5 times compared to the high-order solution with same accuracy for cases with varying Reynolds numbers.

Similarly, Um et al. [162] correct the trajectory of coarse grid simulations to approximate their fine grid counter parts, comparing three different correction approaches for a variety of unsteady problems. They introduce a method, coined solver-in-the-loop, aiming to embed the ML training between the solver iterations using a *differentiable* CFD solver. They conclude that this method provides high accuracy and generalization capabilities with stable long-term corrections. Such differentiable CFD solvers, based on Automatic Differentiation (AD), are traditionally used for aerodynamic shape optimization [81] and allow to compute derivatives of a function with respect to a great number of design variables. In the context of ML, these gradients can be used to optimize the neural network model in an end to end fashion after each solver step, allowing to include the resulting flow field in the training process. Essentially, the differentiable solver enables the computation of derivatives with respect to the trainable ML parameters, ultimately integrating data from the interaction between ML model and CFD solver into the learning process. Such a solver embedded training has been investigated in various other research work to increase the accuracy of coarse grid simulations: one of the earliest work is conducted by Bar-Sinai et al. [7], aiming to replace constant coefficients with predictions of a neural network to improve the spatial derivatives of the Burgers' equation. Kochkov et al. [93] aim to approximate 2D turbulent DNS simulations using a differentiable framework, reporting results for 8-10 times coarser grids with increased speed-ups of up to 80. De Avila Belbute-Peres et al. [41] deploy an adjoint-based differentiable solver in combination with a GNN. The solver is only used on the coarse grid for a fast evaluation of a preliminary solution, which is then up-sampled to the respective fine grid. The up-sampled solution is embedded into the convolutional layers of the GNN model, in turn predicting the final flow field on the fine grid.

However promising such differentiable approaches are, many CFD solvers lack this capability. The adjoint method, a common mode of AD, is difficult to implement and many existing solvers do not easily provide them without significant programming effort. Zhang et al. [180, 181] propose an ensemble Kalman method as an alternative to a differentiable solver to train a neural network. Although the implementation is non-intrusive with regards to the solver, meaning that no changes are needed within the CFD solver, and promising results have been shown for data-driven turbulence modeling, the main disadvantage is that the method does not scale with larger datasets. Another alternative to include indirect data from the CFD solver into the training process is the use of Reinforcement Learning (RL), providing a framework to autonomously optimize specific tasks. In general, a model trained under the RL framework is called an agent, which interacts with an environment. According to its current policy, the agent predicts an action, which induces a transition of the environment's state. Based on the effect of this transition, the agent receives a reward and updates its policy accordingly. This cycle is repeated and the objective is to optimize the agent to receive higher rewards. Because of the flexible formulation of the reward function, RL has recently gained attention in combination with CFD [61, 141]. The reward function allows to embed solver relevant information into the training, without the need of adapting the CFD solver itself. Similarly to adjoints, RL has been explored for shape optimization in fluid problems: Viquerat et al. [164] generate shapes that maximize the lift-to-drag ratio in the flow described by the Navier-Stokes equations at low Reynolds numbers. Similarly, Ghraieb et al. [62] first minimize the drag of a 2D airfoil before applying their methodology to a 3D wing to minimize the lift-to-drag ratio. Keramati et al. [83] use a RL framework to optimize heat exchanger shapes, leading to 60 percent minimized pressure drop and 30 percent maximized heat transfer compared to the initial geometry. Lately, RL has been aimed at improving discretizations: using a multi-agent approach, Novati et al. [125] are able to approximate the energy spectra of DNS data on Large Eddy Simualtions (LES). Similarly, Kurz et al. [101] formulate the reward function to minimize the difference of the energy spectra between LES and DNS. Beck and Kurz [12] investigate the use of RL to find an optimal blending strategy of a hybrid DG and FV scheme. Schwarz et al. [150] use a RL agent to find an optimized slope limiter in a second order FV scheme, where the training is steered by a negative reward to punish nonphysical solutions. Based on these results, RL might be a powerful tool for optimizing certain scenarios without the need for direct modifications to the CFD solver itself, rendering it a potential alternative to differentiable solvers driven by adjoints.

Machine Learning for CFD: Balancing Overly Optimistic and Skeptical Expectations

Summarizing this section, ML methods offer promising avenues, be it as a ROM predicting quantities of interest and thus replacing the solver, or as a correction approximator to enhance simulations, or with data-driven methods interacting with the CFD solver. ML approaches have proven their merit in other fields, and an increased research interest in combining ML with CFD is observable. However, the rising number of publications does not necessarily correlate with success for relevant problems in science and engineering.

Researchers often exhibit skepticism towards ML, known for producing black-box models. Coveney and Highfield [39] describe ML models as "glorified curve-fitting systems". Such skepticism comes from the desire for explainable and reliable models in various scenarios. In contrast, state of the art PDE solvers rely on well established numerical methods that are robust, reproducible, and widely regarded as reliable computational tools. Current ML methods fail to deliver this sense of reliability: Bouthillier et al. [22] highlight the problem of reproducibility of research in the field of deep learning, showcasing that even if results are reproducible, a slight alteration in the experimental set-up would not support the drawn conclusion. While highlighting the opportunities of ML for CFD, the main obstacles of data-driven methods described by Calzolari and Liz [29] are manifold: fast predictions come along with decreased accuracy, databases are often neither sufficiently large nor consistent, and the models lack the capability to extrapolate, while they often over-fit to small scale problems. One can argue that when this last limitation is overlooked in confined test cases, results appear convincing enough for publication, which in turn fuels the optimism for data-driven research. McGreivy and Hakim [120] systematically review a substantial number of articles related to accelerating PDEs for fluid problems with ML, and conclude that mostly positive results are being highlighted, while the negative ones are rarely reported at all. They argue that overoptimism within the ML for CFD community is based on two issues: first, reporting bias leads to the suppression of negative results, and secondly, many positive outcomes are derived by comparing the ML enhanced solution to weak baselines. Take for example the previously mentioned work by Kochkov et al. [93], reporting increased speed-ups of up to 80. When McGreivy and Hakim [120] replicate the problem and compare the efficiency with a stronger baseline, in this case a pseudo-spectral instead of a finite volume method, the ML enhanced methodology is in fact slightly slower.

In conclusion, the current state of ML for CFD is at a crossroad between overly optimistic promises and cautious skepticism. To move forward and build trust in novel tools such as ML within the CFD community, priding itself on accurate and robust methods, thorough research is needed, reporting not only successes, but also addressing limitations. This thesis seeks to contribute to this effort. The aim is to provide a balanced perspective to identify when and how ML methods are truly beneficial and when not, in the context of ML corrections for low-fidelity CFD simulations.

1.3 Objectives and Outline

The central research question of this thesis is outlined below. To address it comprehensively, the results presented in this work are analyzed with respect to three critical properties: robustness, accuracy, and efficiency. Thus, the main research question is broken down into three sub-questions, with each of them targeting one of these key aspects. Furthermore, the investigations are conducted within three different correction scenarios. The sub-questions and corresponding scenarios are listed below.

Main research question
Can ML methods be employed to improve the accuracy of low-fidelity CFD simulations?

Question 1
Can ML models *robustly* infer corrections for low-fidelity simulations under varying conditions?

Question 2
Can the corrected solutions *accurately* approximate the respective high-fidelity ones?

Question 3
Can all of the above be done *efficiently* to maintain the low cost of the low-fidelity simulation?

Correction scenario 1
Correction of steady coarse grid finite volume simulations.

Correction scenario 2
Correction of steady low-order discontinuous Galerkin simulations.

Correction scenario 3
Correction of unsteady low-order discontinuous Galerkin simulations.

These questions are evaluated in this thesis based on the presented scenarios. Chapters 2 and 3 present theoretical background and the proposed methodologies. Chapter 4, 5, and 6 showcase the results for steady coarse grid FV, steady low-order DG, and unsteady low-order DG simulations, respectively. Finally, chapter 7 closes this thesis with a conclusion.

Chapter 2 focuses on the theoretical foundation of this work, beginning with an overview of CFD methodologies, followed by a section on ML. The governing equations of interest are introduced and an overview of the FV and the DG discretization is given. The section concludes with an introduction to temporal discretizations. The ML section describes supervised and reinforcement learning, as well as the different ML models investigated. These include models of varying complexity, namely a random forest, a fully-connected neural network, a graph neural network model, as well as the proximal policy algorithm. Additionally, the section discusses expected generalization capabilities of ML methods and varying approaches of model evaluation and optimization techniques.

Chapter 3 outlines the developed correction methodologies. The supervised training of ML models with the goal to predict a post-processing correction for steady coarse grid FV and low-order DG simulations is described. Secondly, the method is extended to unsteady problems. Additionally, an approach which reduces the variance of the correction term in time is presented, with the aim to provide long-term stable predictions. Finally, an online learning approach employing a RL framework is introduced.

Chapter 4 presents test cases on the post-processing correction of steady coarse grid FV solutions applied to flow conditions involving turbulence and discontinuities. The main goal is to identify how the error induced by a low-fidelity spatial discretization can be quantified and whether this error exhibits patterns which can be learned. The chapter describes the investigated cases, including the 2D RAE2822 airfoil and the 3D LANN wing. The predictive capabilities of a random forest, neural network, and graph neural network are tested and compared.

Chapter 5 continues with the results of the correction of steady low-order DG simulations, extending the previous methodology from FV to another discretization. As opposed to the FV approach, employing a DG discretization allows to generate datasets of varying fidelity on the same grid. The test cases involve turbulent flow around the 2D RAE2822 airfoil, and laminar flow around a 3D delta wing.

Chapter 6 discusses the correction of unsteady low-order DG simulations. Here, different combinations of correction and training approaches are investigated. The first approach, based on the steady methodology, is completely decoupled from the solver, involving supervised learning and a post-processing correction. A more involved approach, aiming to couple ML model and CFD solver during the training and prediction, is based on using RL and applying the correction in-between solver iterations. Investigated problems include the 1D linear advection of a sine curve, and the convection of a 2D isentropic vortex, described by the Euler equations.

Chapter 7 concludes this thesis by reviewing the main objective in the light of the reported results, providing an in-depth discussion and an outlook for future research work.

Chapter 2

Theoretical Background

This chapter introduces the theoretical background of this thesis. Section 2.1 presents various aspects of CFD. This includes the governing equations of interest, the spatial discretization using FV and DG schemes, as well as an overview of temporal discretizations. Section 2.2 focuses on ML topics of this work. This covers two types of learning, supervised and reinforcement learning, and the applied ML models and algorithms, namely random forest, neural network, graph neural network, and proximal policy optimization. Additionally, a discussion on data distribution and its role in the generalization capabilities of ML models is provided. Finally, several validation metrics and hyperparameter optimization techniques are presented, with which ML models are evaluated and fine-tuned.

2.1 Computational Fluid Dynamics

CFD enables engineers and scientists to simulate and explore complex fluid behavior across many applications. In general, the scope of CFD is too broad to be described in this thesis. Thus, the interested reader is referred to common literature for more details [19, 36, 123]. In the following, only aspects relevant for the further content of this work are laid out.

2.1.1 Governing Equations

PDEs are essential to describe physical quantities evolving over space and time. In fluid dynamics, a system of non-linear PDEs captures the conservation laws of mass, momentum, and energy. In general, the governing equations of interest are the so called Navier Stokes equations, given as

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho U) = 0, \tag{2.1}$$

$$\frac{\partial (\rho U)}{\partial t} + \nabla \cdot (\rho U \otimes U) + \nabla P - \nabla \cdot \tau = 0, \tag{2.2}$$

$$\frac{\partial(\rho E)}{\partial t} + \nabla \cdot (\rho H_{tot} U) + \nabla \cdot \boldsymbol{q} - \nabla U : \tau = 0, \tag{2.3}$$

with density ρ, velocity $U = [v_x, v_y, v_z]^T$, pressure P, specific total energy E, and total enthalpy H_{tot}. The conductive heat flux is described by Fourier's law $\boldsymbol{q} = -k_T \nabla T$, with T the temperature and k_T the thermal conductivity. τ denotes the viscous stress tensor, which is given as

$$\tau = \mu_{\text{eff}} \Big[\nabla U + (\nabla U)^T - \frac{2}{3}(\nabla \cdot U)\boldsymbol{I} \Big], \tag{2.4}$$

which takes into account the effective viscosity $\mu_{eff} = \mu_l + \mu_t$, where μ_l and μ_t are the laminar and turbulent viscosity, respectively. For three dimensions, the system consists of five equations and the conservative variables are $\boldsymbol{u} = [\rho, \rho v_x, \rho v_y, \rho v_z, \rho E]^T$. With seven unknown variables, namely ρ, v_x, v_y, v_z, E, P, and T, two additional constitutive equations need to be introduced. Assuming the fluid of interest to behave like a perfect gas, the first constitutive equation to be considered is the ideal gas law

$$P = \rho R T, \tag{2.5}$$

with R being the ideal gas constant. Then, the system of equations is closed with a second constitutive equation, such as Sutherland's law [156], describing the relation between laminar viscosity μ_l and temperature T, with Sutherland's constant C:

$$\mu_L = \mu_0 \frac{T_0 + C}{T + C} \left(\frac{T}{T_0} \right)^{3/2}, \tag{2.6}$$

$$\mu_0 = \frac{\rho_\infty U_\infty L_{ref}}{Re}, \tag{2.7}$$

$$c_p = R \frac{\gamma}{\gamma - 1}, \tag{2.8}$$

where L_{ref} is a reference length, Re the Reynolds number, and c_p the specific heat capacity. γ is the gas dependent ratio of specific heat with $\gamma = 1.4$ for air. Considering laminar flow, the turbulent viscosity is $\mu_t = 0$.

Turbulence Modeling

At a critical Reynolds number, instabilities in laminar flow cause perturbations leading to transition to turbulence [123]. Such turbulent flows are generally chaotic and exhibit a wide range of interacting scales, typically described by Kolmogorov's concept of energy cascade [96, 95]. Resolving the full scale of turbulence, done by so called Direct Numerical Simulations (DNS), is prohibitively expensive and its application has thus been limited to academic problems, including for example flows over a flat plate [173] or turbulent channel flows [111]. To make simulations of turbulent flows feasible at the cost of decreased accuracy, one option to consider is the method of Large Eddy Simulations (LES), which models small scale turbulent structures while resolving the larger ones. The most widely used approach in the aerospace industry, which is also considered within this work, are the Reynolds-Averaged Navier-Stokes (RANS) equations. With the RANS framework, the effects of turbulence are modeled through statistical averaging, rather than directly resolving the turbulent spectrum. This approach requires one or more additional equations for the modeling of turbulence.

Within this thesis, the turbulent viscosity μ_t is obtained using the one-equation Spalart-Allmaras (SA) turbulence model, originally formulated in [155] and extended as a negative formulation in [3]. Thus, one additional equation is introduced for a turbulent transport variable $\tilde{\nu}$, and is given as

$$\frac{\partial \tilde{\nu}}{\partial t} + \nabla \cdot (\tilde{\nu} U) = \frac{1}{\sigma}\left[\nabla \cdot ((\nu + \tilde{\nu})\nabla \tilde{\nu}) + c_{b2}(\nabla \tilde{\nu})^2\right] + P_t - D_t, \tag{2.9}$$

where $\nu = \mu_\mathrm{L}/\rho$ is the molecular viscosity, and P_t and D_t denote the production and destruction term, respectively. This work neglects the trip term T_t, introduced in the original SA formulation. Thus, the vector of conservative variables is extended to $\boldsymbol{u} = [\rho, \rho v_x, \rho v_y, \rho v_z, \rho E, \rho\tilde{\nu}]^T$. Finally, the turbulent transport variable $\tilde{\nu}$ is linked to the eddy viscosity μ_t as

$$\mu_t = \rho \tilde{\nu} f_{v1}. \tag{2.10}$$

For the complete definition of the SA equation, including the constants and other terms, such as σ, c_{b2}, and f_{v1}, the reader is referred to [3, 155].

As resolving turbulent effects is still mostly infeasible for practical applications, research on turbulence modeling is still an active area. Today's turbulence models are often specialized, tailored to specific flow regimes or applications, with no model being universally reliable among many scenarios. Additionally, each model involves trade-offs between accuracy and computational cost. An in-depth discussion of the field of turbulence modeling is beyond the scope of this work, and for details the reader is referred to common literature [170].

Euler Equations

The governing equations described in (2.1), (2.2), and (2.3) can be simplified by neglecting certain components. In high Reynolds number flows, viscous effects can be confined to thin boundary layers near solid surfaces and are negligible in the majority of the flow domain. Under these conditions, the viscous terms in the momentum and energy equations, given in (2.2) and (2.3), vanish. This leads to the so-called Euler equations, describing the convection of inviscid fluids. Even if they represent a simplified form, the Euler equations allow to represent important phenomena such as shocks, relevant for aerospace applications.

Linear Advection Equation

The phenomenon of advection describes the transport of a conserved quantity or scalar field $u(x, t)$ induced by fluid motion. The PDE representing this linear advection, where the transport takes place along the x-axis in 1D, is given by

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \tag{2.11}$$

with c a constant non-zero velocity coefficient.

2.1.2 Spatial Discretization

Most approaches approximating the governing equations follow two steps prior to the temporal discretization: first of all, the bounded physical domain of interest Ω is transformed into a computational domain, on which the solution $\boldsymbol{u}$ is approximated as $\tilde{\boldsymbol{u}}$ [19]. For this, consider the bounded computational domain Ω_h covered by a finite set of domains $\{k_i\}_{i=1,...,I}$. Thus, the set

$$\Omega_h := \{k_i : i = 1, ..., I\} \tag{2.12}$$

comprises I non-overlapping elements or control volumes k. This decomposition Ω_h is the so-called grid or mesh. The second step is the discretization of the spatial terms of the boundary value problem of interest, by, for example, the FV or the DG method. Although the FV discretization is more commonly used, the DG discretization is introduced first, as it inherently includes the FV scheme in its simplest form.

Discontinuous Galerkin Method

The DG discretization receives currently more attention from academia than from industry, as high-order methods have so far not been applied to an extensive range of industrial applications since many dismiss it for being too expensive [20, 168]. The main promise of higher order methods, including DG, is to reach improved accuracy with fewer degrees of freedom compared to methods like FV. Additionally, the DG formulation offers great flexibility in terms of numbers of unknowns, as the number of degrees of freedom can be steered by both h- and p-refinement. Thus, there is still interest in developing DG solvers [50, 159].

To discretize the governing equations with a DG scheme, consider the functional space V_h^p of piecewise polynomials up to degree p on all elements k on the grid Ω_h, given as

$$V_h^p = \{\phi \in L^2(\Omega_h) : \phi|_k \in V_k^p, \forall k \in \Omega_h\}. \tag{2.13}$$

Here, V_k^p defines the space of polynomials of total degree p on element k. A basis of V_k^p is defined as $\phi_k = \{\phi_k^1, ..., \phi_k^N\} \in V_k^p$, with dimension N. The numerical solution $\tilde{\boldsymbol{u}}$ is expressed in each element k as piecewise polynomial function

$$\tilde{\boldsymbol{u}}(x,t) = \sum_{i=1}^{N} a_i(t)\boldsymbol{\phi}_i(x), \qquad \forall x \in k, \quad k \in \Omega_h, \quad \forall t > 0, \tag{2.14}$$

with the polynomial coefficients $\boldsymbol{a} = [a_1, \ldots, a_N]$ being the degrees of freedom per equation per element k. In this work, a hierarchical and orthonormal basis function is employed, defined by a modified Gram-Schmidt procedure as proposed in [8]. Neglecting the time dependent term of the governing equations, the weak formulation using the DG discretization can be written as

$$\langle \phi, \nabla \cdot F_c(\tilde{\boldsymbol{u}}) \rangle_k - \langle \phi, \nabla \cdot F_v(\tilde{\boldsymbol{u}}, \nabla \tilde{\boldsymbol{u}}) \rangle_k = 0, \tag{2.15}$$

with $\langle q, w \rangle_k$ indicating the scalar product of q and w over element k and $F_c(\tilde{\boldsymbol{u}})$ and $F_v(\tilde{\boldsymbol{u}}, \nabla \tilde{\boldsymbol{u}})$ being the convective and diffusive fluxes, respectively. Integration by parts yields

$$\begin{aligned} &\langle \nabla \phi, F_c(\tilde{\boldsymbol{u}}) - F_v(\tilde{\boldsymbol{u}}, \nabla \tilde{\boldsymbol{u}}) \rangle_k \\ &-\langle \phi, \nabla \cdot F_c(\tilde{\boldsymbol{u}}) \cdot \boldsymbol{n} \rangle_{\partial k} + \langle \phi, \nabla \cdot F_v(\tilde{\boldsymbol{u}}, \nabla \tilde{\boldsymbol{u}}) \cdot \boldsymbol{n} \rangle_{\partial k} = 0. \end{aligned} \tag{2.16}$$

The number of degrees of freedom N is related to the problem dimension d and the polynomial degree p within each element, as given in (2.17). For instance, for a 3D problem, the number of degrees of freedom can be obtained with (2.18).

$$N = \frac{(p+d)!}{p!d!} \tag{2.17}$$

$$N_{\mathrm{3D}} = \frac{(p+1)(p+2)(p+3)}{6} \tag{2.18}$$

To display how quickly the number of degrees of freedom N rises with increased polynomial degree p, they are exemplarily given in table 2.1 for $p \in \{0, 1, 2, 3, 4, 5\}$ for dimensions $d = \{1, 2, 3\}$. Taking into consideration that solving the RANS equations scales with $\mathcal{O}(N^m)$, with $m \geq 2$, it becomes clear why higher order methods are often dismissed for being too expensive.

Table 2.1: Number of degrees of freedom N per element and equation for varying polynomial degrees p and dimensions d

d	$p=0$	$p=1$	$p=2$	$p=3$	$p=4$	$p=5$
1	1	2	3	4	5	6
2	1	3	6	10	15	21
3	1	4	10	20	35	56

As the name of the discretization method suggests, this formulation leads to discontinuities at element boundaries. To overcome this, numerical flux schemes are introduced to compute convective fluxes at element boundaries. Furthermore, the viscous term needs special attention due to gradients, for which varying approaches exist, including Symmetric Interior Penalty (SIP) [70], as well as first and second Bassi-Rebay scheme (BR1 [9] and BR2 [10]), allowing to evaluate a diffusive flux at element boundaries. Another aspect which has to be taken into account is the treatment of discontinuities, specifically shocks, since higher polynomial degrees lead to Gibbs-type oscillations, reducing the robustness of the scheme significantly.

In this work, if not stated differently, the Roe upwind scheme [19] and, in the case of viscous flows, the BR2 scheme [10] are employed to compute convective and diffusive numerical fluxes at element boundaries, respectively. In the presence of shocks, a sensor-based artificial viscosity method is employed [137]. If not stated differently, the DG simulations in this work are conducted using the CFD software developed by ONERA, DLR, Airbus (CODA) [165].

Finite Volume Method

The FV method is a widely used scheme for the spatial discretization of PDEs, especially for fluid dynamics and heat transfer [172]. For many applications nowadays, its second order accurate form results in an acceptable trade-off between accuracy and cost. Additionally, it can be argued that the method is relatively straightforward to implement [123]. For details and the derivation of the FV method, the reader is referred to further literature [19, 123].

For the scope of this thesis, it is interesting to highlight the relationship between DG and FV schemes. Consider (2.14) and the lowest possible polynomial degree $p = 0$, such that $N = 1$ and $\phi_1 = 1$. With this, the numerical solution $\tilde{u}$ is simply a constant value across each element K, which is also the case for a first order FV discretization. By introducing gradient reconstruction methods, such as the Green-Gauss reconstruction, does its popular second order formulation achieve improved accuracy.

In general, using the FV method, solutions of the boundary value problem of interest can be obtained at discrete points of the grid, either in the barycenter of the control volumes K following a cell-centered scheme, or, as is done in this work, in the element's vertices, employing a node-centered scheme. Additionally, for the FV simulations conducted within this thesis, a pressure switch in the dissipative term ensures robustness in cases featuring shocks by inducing first order accuracy along the shock location [106]. All FV simulations in this work are conducted using the DLR CFD solver TAU [98],

2.1.3 Temporal Discretization

Assuming steady-state flow, the time dependent terms in the governing equations can be neglected. This can obviously not be done for simulations of time dependent phenomena, like gust responses, vibrations, and flutter, where the solution changes over time. Thus, the terms need to be further integrated over a finite time step Δt and methods are required to evaluate these integrals from the current time step t_i to $t_{i+1} = t_i + \Delta t$. A common approach handling spatial and temporal discretization independently is the method of lines, allowing for a flexible choice of the respective schemes [19]. Here, the governing equations are firstly discretized using a spatial discretization such as FV or DG. This results in a system of ordinary differential equations in time, which can then be integrated using a time-stepping scheme. Such temporal schemes are usually categorized as explicit or implicit. Explicit schemes on the one hand compute the solution at the next time step based on information only available at the current time step. These methods are comparably simple to implement and computationally inexpensive per time step. However, they usually require small time steps to ensure stability, which poses a major restriction for stiff equations. Implicit methods on the other hand calculate the next time step taking into account both current and future state information. This renders implicit methods more robust, allowing for greater time step sizes.

For unsteady simulations in this work, a third order explicit Runge-Kutta (RK) method [19] is employed for simple 1D problems. For more complex problems, a dual time stepping scheme with linearized implicit Euler is employed for the inner iterations and a diagonally implicit RK scheme for the outer iterations.

2.2 Machine Learning

At its core, ML encompasses a set of methods designed to detect patterns in data during the so-called training, without being explicitly programmed to do so. Once trained, the ML model or function approximator leverages these patterns to make new predictions or decisions. Depending on the nature of the task, the function approximator can be a classifier predicting qualitative outputs, or, as it is the case for this work learning corrections for flow field variables, a regressor predicting quantitative outputs [71].

As for the previous section, the field of ML is too vast to be presented in depth, and the interested reader is referred to common literature for more details [18, 71, 124]. In the following, section 2.2.1 gives an introduction to supervised and reinforcement learning, while section 2.2.2 presents the ML models and algorithms employed within this work. A discussion on the generalization capabilities of trained models is given in section 2.2.3, emphasizing the importance of coherent data distribution. In section 2.2.4, validation metrics, which aid in evaluating and comparing models, are described and the concept of hyperparameter optimization used to fine-tune models is introduced.

2.2.1 Categories of Learning

ML methods are typically grouped into supervised learning, unsupervised learning, and reinforcement learning [71, 124]. Supervised learning owes its name to the fact that labeled data is needed to train models to map inputs to the respective outputs. Unsupervised learning aims at uncovering patterns within unlabeled data, creating for instance models to find groups or clusters within a dataset. Finally, reinforcement learning involves learning to make decisions by interacting with an environment and receiving feedback, known as reward, based on the previously taken actions. This work is centered around two of these categories, namely supervised learning and reinforcement learning.

Supervised Learning

Since supervised learning relies on labeled data, it requires pairing each input with its corresponding output manually, a process that can become labor-intensive when working with vast datasets. Denoting the inputs, also defined as features or independent variables, as $\boldsymbol{\eta} = \{\eta_1, ..., \eta_J\}$, and the corresponding outputs, also known as responses or dependent variables, as $\boldsymbol{y} = \{y_1, ..., y_J\}$, and assuming a relation between the J input and output pairs or instances, the aim of the training is to find a function approximator defined as

$$f_{ML}(\boldsymbol{\eta}, \theta) = \hat{\boldsymbol{y}}, \tag{2.19}$$

where θ are the parameters of the model and $\hat{\boldsymbol{y}}$ is the predicted output. The training involves the search for parameters θ of the function approximator. In addition to the model parameters θ, there are so-called hyperparameters, which are settings that guide the training process and define the model architecture. These hyperparameters have to be defined before the training, either manually or through a hyperparameter optimization process described later in section 2.2.4. The training, and thus the search for parameters θ, is driven by minimizing a loss function $\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$, oftentimes the mean squared error, which measures the distance between prediction $\hat{\boldsymbol{y}}$ and the ground truth value $\boldsymbol{y}$.

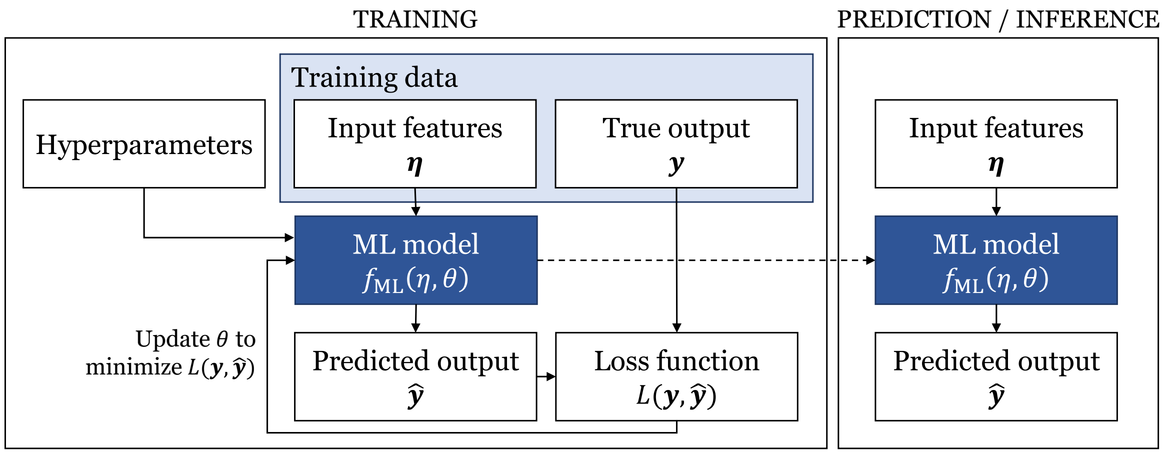

TRAINING
PREDICTION / INFERENCE
Training data
Hyperparameters
Input features $\boldsymbol{\eta}$
True output $\boldsymbol{y}$
ML model $f_{\mathrm{ML}}(\eta, \theta)$
Update θ to minimize $L(\boldsymbol{y}, \hat{\boldsymbol{y}})$
Predicted output $\hat{\boldsymbol{y}}$
Loss function $L(\boldsymbol{y}, \hat{\boldsymbol{y}})$
Input features $\boldsymbol{\eta}$
ML model $f_{\mathrm{ML}}(\eta, \theta)$
Predicted output $\hat{\boldsymbol{y}}$

Figure 2.1: Supervised learning

Figure 2.1 depicts the training and prediction phase of a supervised learning model. During the training phase, the input features of the training dataset are passed to the ML model, which predicts based on current parameters θ an output $\hat{\boldsymbol{y}}$. The prediction $\hat{\boldsymbol{y}}$ and the true output $\boldsymbol{y}$ from the training data are used to compute the loss function $\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$, which in turn drives the update of the model parameters θ. This process is repeated until convergence of the loss function or another stopping criterion is reached. The trained ML model with fixed parameters θ is then employed during the prediction or inference phase on new input features, which have not been encountered in the training data.

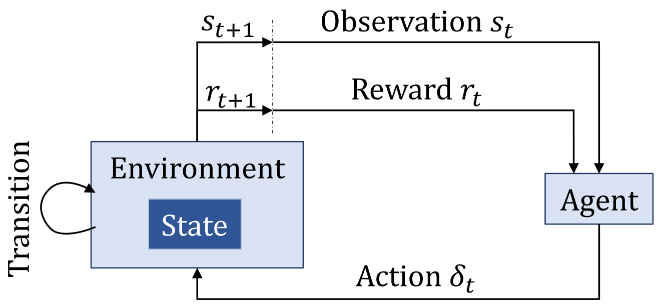


Figure 2.2: Interaction between environment and agent in reinforcement learning

Obviously, being based on statistical learning and optimization techniques, an ML prediction $\hat{\boldsymbol{y}}$ will in general not match the true output $\boldsymbol{y}$ for several reasons. These include noise in the training data, incomplete inputs not fully representing the underlying pattern to represent the output, limitations of the chosen model, and sub-optimal training leading to local optima for the ML parameters θ. Nevertheless, through careful model selection, feature engineering, and iterative training processes, predictions with sufficient accuracy can be achieved for practical applications.

Reinforcement Learning

Opposed to supervised learning, RL does not learn a function approximation based on input and output pairs, aiming to reduce a loss function. Instead, the learning is based on an interaction with a so-called environment, and the goal is to maximize the expected return [157]. The key concept of RL is presented in Figure 2.2. The agent, oftentimes one or multiple ML models, receives the state s_t of the environment at time t. This state does not necessarily reflect the complete state describing the environment, but can rather be only a partial observation of it. Based on the current policy π of the agent, it returns an action δ_t to the environment given the current state s_t. The action induces a transition to the next environment state s_{t+1}. During training, the agent receives a reward r_{t+1} and the new state s_{t+1}. This procedure is repeated, until the terminal state s_n is reached. A complete trajectory, from initial state s_0 to a terminal state s_n leading to rewards r_1 to r_{n-1}, is called an episode. Employing a stochastic policy, the agent's goal is to find over multiple episodes a policy $\pi_\theta(\delta|s = s_t)$ which maps the states s to the probabilities of selecting each possible action δ, maximizing the cumulative reward over time, also called return G_t. The return G_t for a infinite episode is defined as

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2.20}$$

with $0 \leq \gamma \leq 1$ being the discount rate. This parameter balances the importance of current and future rewards and acts as a regularization term, with $\gamma = 0$

trying to maximize only the immediate reward r_{t+1}. With γ approaching 1, future rewards become more relevant, increasing the importance of long-term actions [4].

The interaction between agent and environment in RL underlies the assumption that the learning task can be modeled as a Markov Decision Process (MDP), usually used to model sequential decisions under uncertainty. Thus, the RL task needs to satisfy the Markov property [157]. Consider the transition probability

$$p_{tr}(s_t, r_t | s_{t-1}, \delta_{t-1}), \tag{2.21}$$

which defines the probability of each next possible state s_t and reward r_t pair, given state s_{t-1} and action δ_{t-1} pair from the previous time step. With these definitions, a finite MDP can formally be defined as

$$\mathrm{MDP} = (s, \delta, p_{tr}, r, \gamma, H), \tag{2.22}$$

with s the set of all possible states, δ the set of possible actions, p_{tr} the transition probability, r the set of rewards, γ the discount factor, and H the horizon of the finite sequence. Given an environment that satisfies the Markov property, the next state and expected reward are predictable solely using the current state and action, and the best policy to choose an action based on the current state is as good as the best policy on the basis of not only the current state, but the complete history of past states. If the agent cannot observe the complete state of the environment, but only a fraction of it, one usually defines the system as a Partially Observable MDP (POMDP) [78], which is indeed the case for most practical applications.

Further foundations of modern RL are based on studies of learning by trial and error, previously researched primarily in animals [67], and on the work of Richard Bellman on dynamic programming [15]. Dynamic programming is a field which strives to find algorithms to solve complex problems by breaking them into sub-problems and solving them recursively. Bellman et al. [14] introduced MDPs for optimal control problems and developed a formal approach to solve them using the state of a dynamical system and an optimal return function, which is nowadays known as the Bellman equation. Using the Bellman equation for RL, the so-called value function is defined as

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[G_t \mid s_t = s] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right] \\ &= \mathbb{E}_\pi\left[r_{t+1} + \gamma G_{t+1} \mid s_t = s\right] \end{aligned} \tag{2.23}$$

with $\mathbb{E}_\pi[... \mid s_t = s]$ denoting the expected value for an agent following the policy π given state s_t. Estimating the value function is central to almost all RL algorithms [157]. The value function defines how good it is to be in a given state s_t in terms of the expected return G_t under policy π, thus it is also known as the state-value function of the policy. Similarly, the Bellman equation for the action-value function or the Q-function under policy π is defined as

$$\begin{aligned} Q_\pi(s,\delta) &= \mathbb{E}_\pi[G_t \mid s_t = s, \delta_t = \delta] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, \delta_t = \delta\right] \\ &= \mathbb{E}_\pi\left[r_{t+1} + \gamma G_{t+1} \mid s_t = s, \delta_t = \delta\right]. \end{aligned} \tag{2.24}$$

The Q-function defines the value for the expected return when taking action δ given state s under policy π. Thus, the Bellman equation expresses recursively a combined value of current and possible successor states for the value function $V_\pi(s)$ and a value for the current and possible successor state and action pairs for the Q-function $Q_\pi(s,\delta)$. Based on (2.23) and (2.24), the advantage function is defined as

$$A_\pi(s,\delta) = Q_\pi(s,\delta) - V_\pi(s), \tag{2.25}$$

which measures the difference between the action-value function and the state-value function under policy π. It describes how much better it is to choose the new action δ compared to following the current policy π, i.e. the advantage of taking action δ instead of the default action of policy π.

Although RL is not a new concept, it has regained popularity due to the integration of Neural Networks (NNs) as function approximators, coining the term Deep Reinforcement Learning (DRL). By modeling the value function or the policy with NNs, high-dimensional state and action spaces can be handled more efficiently. RL shares similarities with other optimization algorithms, since it tries to find an optimum by exploring and exploiting the interactions with the environment. Compared to classical optimization algorithms, the use of NNs in DRL enables the trained model to be extended to new tasks, making DRL a more versatile tool applicable to large scale, complex, and high-dimensional environments. Nevertheless, a key limitation of DRL is the inherent stochastic nature of the method, making the training and results highly variable. This randomness can lead to inconsistent performance across tasks, making the algorithms sensitive to implementation details [48]. This and the fact that small changes in hyperparameters or environmental factors can drastically affect outcomes, renders DRL difficult to reproduce. In the following parts of this work, the abbreviation RL will be used interchangeably with DRL.

In summary, the ability to learn directly from past interactions with an environment makes RL a unique tool for dynamic and time dependent tasks. Nevertheless, RL is a vast field and, compared to supervised learning, has not yet found broad usage due to its added complexity. Nowadays, many libraries exist, offering a variety of RL algorithms out of the box [44, 99, 139]. Nevertheless, one fundamental building block in RL is ultimately problem specific: casting the environment into an MDP, often requiring to implement algorithms from scratch. Defining an environment involves defining the observable state, transition function, and the shape of the reward function, as well as balancing short- and long-term returns. Especially the design of the reward function is crucial, since poorly shaped rewards can lead to sub-optimal policies and thus to unintended behavior. Additionally, many RL algorithms exist. The choice of a suitable algorithm is dependent on the problem at hand, since some handle only discrete action or state spaces, while others can handle continuous data. What distinguishes these algorithms furthermore is how sample-efficient, computationally expensive, and stable they are. For example, some sample-efficient and thus less expensive off-policy algorithms include a memory replay buffer, in which past experienced episodes are stored in, such that they can be redrawn and reused for training [115]. Thus, finding problems for which RL is relevant requires not only understanding its potential, but also navigating its many challenges.

2.2.2 Machine Learning Models and Algorithms

A broad range of ML models exist, varying in their complexity and learning algorithms. Simpler models, such as linear regression or decision trees, are interpretable and require less computational power, while more complex models, such as deep learning models, can capture intricate patterns in data but are often considered black-boxes and require more resources. The models also differ in their ability to handle different types of data, how prone they are to overfitting, and the effort required to find optimal hyperparameters. Thus, the choice of model depends on multiple factors, including the complexity of the problem, the amount of data available, and a balance between available resources and expected performance.

The following sections introduce random forest, neural network, and graph neural network models. These models are chosen for this work based on their capability of handling large datasets and capturing non-linear relationships effectively. Random forests are particularly popular due to the ease of tuning and training them, while achieving high accuracy across a variety of tasks without extensive parameter optimization or the need for data scaling [71, 133]. Deep learning methods, with neural networks at their core, have perpetually gained attention in many fields, due to their ability of handling big datasets and a vast range of complex tasks [35]. Both random forests and neural networks have been explored in previous work related to CFD discretization errors [30, 69, 110], rendering their selection a reasonable choice. Furthermore, the use of graph neural networks is explored, since they are designed to operate on graph structured data, which is typically encountered in CFD simulations [74, 122].

Additionally, this section describes the Proximal Policy Optimization (PPO) RL algorithm, which uses neural networks as function approximators to model policy and value functions. PPO can handle both discrete and continuous state and action spaces, and is known to be relatively easy to implement while the training is more stable compared to other RL algorithms. Despite its simplicity, PPO often matches or exceeds the performance of more complex RL algorithms, and has become one of the go-to algorithms for many RL tasks [48, 178].

Random Forest

Random Forests (RF) were first introduced by Breiman [25] and belong to the so-called ensemble methods. Creating an ensemble model is based on the principle that a collective decision yields in general improved results compared to the individual ones. Thus, employing ensembles constitutes of combining the results of two or more base models or weak learners, which may individually perform poorly [71, 182]. The RF consists of the set $\{T(\boldsymbol{\eta}; \Theta_i)\}_{i=1}^{D}$ of D decision trees. Here, $\boldsymbol{\eta}$ denotes the feature inputs and Θ_i the random characteristics, or random vector, leading to the i-th decision tree. The random vector includes the selected features for a split, the cut point for a split at each node, and the values of terminal nodes. After B trees are grown, the final regression result $\hat{y}_{\mathrm{RF}}(\boldsymbol{\eta})$ of the RF is the averaged output of all D decision trees in the ensemble set $\{T_i\}_{i=1}^{D}$:

$$\hat{\boldsymbol{y}}_{\mathrm{RF}} = \frac{1}{D} \sum_{i=1}^{D} T_i(\boldsymbol{\eta}; \Theta_i)). \tag{2.26}$$

Improved performance is achieved if the weak learners exhibit diversity among themselves [100]. For RF, this is achieved firstly via a random selection of feature variables considered for the splitting of the tree nodes, and secondly via bootstrap aggregation, also known as bagging. Bootstrapping is a sampling technique, in which random subsets with replacement are drawn from the entire dataset. This results in diverse subsets and thus diverse weak learners, improved performance on limited datasets, and reduced overfitting. Using decision trees as weak learners allows to decrease the bias by increasing their depth. Additionally, averaging the results of the weak models reduces the variance of the final prediction [71]. The model parameters which are adapted to optimize the loss function during the training are the selected features and the threshold for a split of the respective tree node. Summarized, the RF training algorithm for a regression task, driven by minimizing a loss function $\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}_{\mathrm{RF}})$, is given in algorithm 1. In this work, RF models are built using the Scikit-learn [135] library.

Algorithm 1 Random Forest Training

```
for $i = 1$ to $D$ do
    1) Bootstrap: draw a subset with replacement from training data
    2) Grow tree on subset:
       Recursively repeat these steps until a stopping criterion is reached
          a. Randomly select a subset of all feature variables
          b. Select feature for split resulting in greatest loss reduction
          c. Split node into two child nodes
end for
Return ensemble of trees $\{T_i\}_{i=1}^{D}$
```

Neural Network

Neural Networks (NN), inspired by the brain's neural structure, build the core of deep learning [63]. They have become a versatile and powerful tool due to their ability to learn complex, high-dimensional, and non-linear relationships. First introduced in 1943 [119], NNs are nowadays the foundation of a range of advanced deep learning architectures and algorithms, including convolutional neural networks [108], deep reinforcement learning [157], or large language models [121]. Thus, understanding their fundamental concept is of relevance for further architectures investigated in this work. A NN consists of layers of so-called neurons, whereas for the l-th layer the output h^l is described as

$$h^l = g(\boldsymbol{W}^l h^{l-1} + \boldsymbol{b}^l), \tag{2.27}$$

with $1 \leq l \leq L$, where h^{l-1} is the output of the previous layer, and $h^1 = \boldsymbol{\eta}$. g is a non-linear activation function, and the model parameters θ, which are adapted during training to optimize a loss function, are $\boldsymbol{W}$, the weight matrix, and $\boldsymbol{b}$, the bias vector. The shape of the weight matrix is defined by the number of inputs and the number of neurons, and the bias vector contains one bias per neuron. Thus, the final prediction $\hat{\boldsymbol{y}}_{NN}$ of an NN comprising L layers is written as

$$\hat{\boldsymbol{y}}_{NN} = g(\boldsymbol{W}^L h^{L-1} + \boldsymbol{b}^L). \tag{2.28}$$

The training of an NN aims to find optimal parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ to minimize a loss function $\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}_{NN})$, as summarized in algorithm 2. This is done iteratively over so called epochs. During each epoch, the data is shuffled and divided into batches. For each batch i, gradients of the loss function with respect to the weights and biases are calculated, i.e. $\frac{\partial \mathcal{L}_i}{\partial \boldsymbol{W}}$ and $\frac{\partial \mathcal{L}_i}{\partial \boldsymbol{b}}$. The algorithm computing these gradients, starting from the output at the end of the network and moving to the initial layers, is called backpropagation. The computed gradients finally drive an optimization algorithm, commonly stochastic gradient descent [143] or Adam [88], to adjust the parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ based on a learning rate, which influences convergence and stability. This process continues over multiple epochs until a certain stopping criterion is reached. For the NN model employed in this work, the PyTorch [134] library is used.

Graph Neural Network

Graph Neural Networks (GNNs) have been proposed as an extension of NNs for graph structured or so-called non-Euclidean data [147]. The advantage of using GNNs is twofold: firstly, they are applicable to graphs of arbitrary structure, since they are permutational invariant to the number of input nodes and edges, while traditional neural network architectures typically are constrained to fixed-size data with regular structures, as is encountered for example on images. Secondly, they incorporate the topology of the graph, such that their predictions consider local neighborhood information and thus the relation between data points. These characteristics make GNNs highly relevant for CFD, since CFD data is often obtained

Algorithm 2 Neural Network Training

```
Initialize weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$ of NN
for epoch $e$ in $E$ or until stopping criterion is reached do
    Shuffle training data $(\boldsymbol{\eta}, \boldsymbol{y})$
    Divide data into $B$ batches
    for batch $i$ in $B$ do
        Forward propagation: compute prediction $\hat{\boldsymbol{y}}_{NN,i}$ for feature input $\boldsymbol{\eta}_i$
        Compute loss $\mathcal{L}_i = \mathcal{L}(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_{NN,i})$
        Backpropagation: compute gradients $\frac{\partial \mathcal{L}_i}{\partial \boldsymbol{W}}$ and $\frac{\partial \mathcal{L}_i}{\partial \boldsymbol{b}}$
        Update parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ using an optimization algorithm
    end for
end for
Return NN model
```

from unstructured grids and can be easily represented as a graph. Moreover, the solution in one grid element is naturally influenced by a stencil of surrounding elements.

Consider a graph $\mathcal{G} = (V, \mathcal{E})$, defined by a set of vertices or nodes V and a set of edges $\mathcal{E}$. Any vertex v_k is connected to any vertex $v_j \in \mathcal{N}(v_k)$ through the edges $e_{j,k}$, where $\mathcal{N}(v_k)$ is the set of all neighborhood vertices of v_k. The core principle of GNNs is to update the initial feature vector representation of v_k given as $\boldsymbol{h}_k^{t=0} = \boldsymbol{\eta}_k$ to a new feature vector representation $\boldsymbol{h}_k^{t+1}$. Figure 2.3 depicts an undirected graph with 6 vertices, with each vertex having as attribute a feature vector $\boldsymbol{\eta}_v$, making up in their entirety the current graph feature representation $\boldsymbol{h}^{t=0}$. Note that the edges are only exemplarily indicated for $k = 0$. The update to $\boldsymbol{h}_k^{t+1}$ is dependent on the feature vector of the current vertex at previous step $\boldsymbol{h}_k^t$, the feature vectors of all neighboring vertices $\boldsymbol{h}_j^t$ for all $v_j \in \mathcal{N}(v_k)$, and if applicable any edge feature vectors.

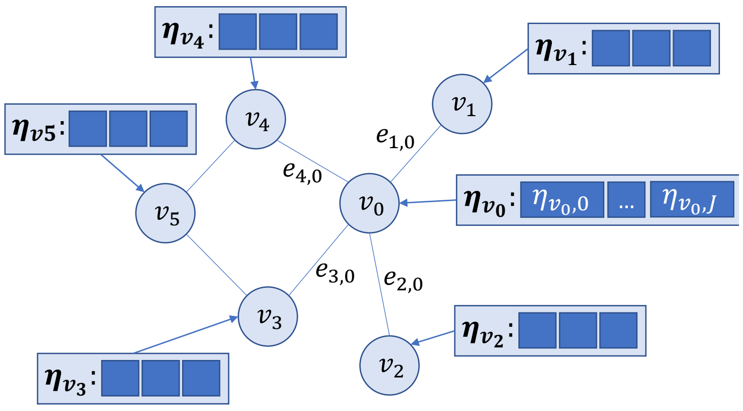

$\boldsymbol{\eta}_{v_4}$:
$\boldsymbol{\eta}_{v_1}$:
$\boldsymbol{\eta}_{v5}$:
v_4
v_1
$e_{1,0}$
$e_{4,0}$
v_5
v_0
$\boldsymbol{\eta}_{v_0}$: $\eta_{v_0,0}$ ... $\eta_{v_0,J}$
$e_{3,0}$
$e_{2,0}$
v_3
$\boldsymbol{\eta}_{v_2}$:
v_2
$\boldsymbol{\eta}_{v_3}$:

Figure 2.3: Undirected and attributed graph

This update is done in three steps: message passing, aggregation, and the final up-

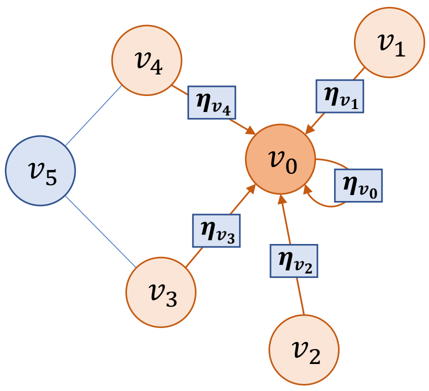


Figure 2.4: Message passing to node $v_k = v_0$

date of the feature vector representation. During the message passing, a function M_t, which entails model parameters to be optimized during training, passes neighborhood information as a message $\boldsymbol{m}_{j,k}^{t+1}$ from each neighbor vertex $v_j \in \mathcal{N}(v_k)$ to vertex v_k, as given in (2.29). Figure 2.4 shows the message passing for vertex $v_k = v_0$, propagating information from immediate neighbors and the vertex's own feature vector. With this, each additional graph convolutional layer increases the radius within which information is propagated. Thus, one layer aggregates information from immediate neighboring vertices, whereas two layers aggregate information including the neighbors of neighbors, and so on. During the aggregation step, all received messages need to be combined into the message $\boldsymbol{m}_k^{t+1}$. This aggregation function $\bigoplus$ commonly consists of a sum or mean of all received messages, such that the final message is independent of the number of neighbors or their indices and thus permutation-invariant. The general form of the aggregation function is given in (2.30). In the final step, the feature vector representation of vertex v_k is updated based on a function U_t, which, similarly to the message passing function, contains model parameters that are to be optimized during training. This update function U_t takes into account the aggregated message $\boldsymbol{m}_k^{t+1}$ and the previous feature vector representation $\boldsymbol{h}_k^t$, as given in (2.31). Combining these three steps, i.e. message passing, aggregation, and feature representation update, yields (2.32).

$$\boldsymbol{m}_{j,k}^{t+1} = M_t(\boldsymbol{h}_j^t, \boldsymbol{h}_k^t, \boldsymbol{e}_{j,k}) \tag{2.29}$$

$$\boldsymbol{m}_k^{t+1} = \bigoplus_{v_j \in \mathcal{N}(v_k)} \boldsymbol{m}_{j,k}^{t+1} \tag{2.30}$$

$$\boldsymbol{h}_k^{t+1} = U_t(\boldsymbol{h}_k^t, \boldsymbol{m}_k^{t+1}) \tag{2.31}$$

$$\boldsymbol{h}_k^{t+1} = U_t\left(\boldsymbol{h}_k^t, \bigoplus_{v_j \in \mathcal{N}(v_k)} M_t(\boldsymbol{h}_j^t, \boldsymbol{h}_k^t, \boldsymbol{e}_{j,k})\right) \tag{2.32}$$

Since the message passing and aggregation steps can be defined in different ways, many variations of GNN layers exist. These layers can be nested, thus repeating

(2.32), which ultimately increases the aggregation radius and the number of model parameters. This work employs the Graph Convolutional Network (GCN) layer proposed in [89] as well as the residual gated graph network proposed in [26]. Exemplarily shown for the residual gated graph network, the convolutional operator is given as

$$h_k^{k+1} = \sigma(U_t h_k^t + \sum_{v_j \in \mathcal{N}(v_k)} \lambda_{kj} \odot M_t h_j^t). \tag{2.33}$$

Here, σ is a non-linear activation function, and $\odot$ signifies the point-wise Hadamard multiplication operator. The first term in brackets represents the self-update of node k, transforming the node's own features. The second term captures the aggregation of information from neighboring nodes j, where $j \to k$ denotes that node j is a neighbor of node k. Here, λ_{kj} is a weighting term, with the Sigmoid function as non-linear activation σ, while containing further trainable terms A_t and B_t, such that

$$\lambda_{kj} = \sigma(A_t h_k^t + B_t h_j^t). \tag{2.34}$$

GNNs are trained like NNs, as summarized in algorithm 2: at each epoch, data is divided into batches and the predicted output per batch is used to compute a loss function. This loss is differentiated via backpropagation with respect to the model parameters, which in turn drives an optimizer to update the model parameters to minimize the loss function. For all GNN models in this work, the PyTorch Geometric [53] library is employed.

Proximal Policy Optimization

The PPO algorithm was first introduced in 2017, with the aim of improving the scalability, data efficiency, and robustness of leading RL algorithms at that time [149]. As an on-policy method, PPO updates its policy using samples generated from the current policy, as opposed to off-policy methods, which are trained on past experiences. While this approach simplifies the learning process, it makes PPO less sample efficient compared to other off-policy algorithms, since experiences from previous policies are not reused.

PPO belongs to the family of Policy Gradient (PG) methods, which focus on directly optimizing the policy. In essence, PG methods estimate the policy gradient and utilize a stochastic gradient ascent algorithm to improve the policy iteratively, using a gradient estimator of the common form

$$\hat{q} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(\delta_t | s_t) \hat{A}_t \right], \tag{2.35}$$

where the expectation $\hat{\mathbb{E}}_t[...]$ is collected empirically and averaged over a batch of samples. θ denotes the policy model parameters to be optimized, and $\hat{A}_t$ is an estimator of the advantage function at time step t. Using automatic differentiation, as it is common with the training of NNs, the gradient estimator is obtained by differentiating the objective function

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(\delta_t | s_t) \hat{A}_t \right]. \tag{2.36}$$

It was shown empirically that using this formulation leads to large updates of the policy network, leading to unstable learning. To avoid this, the authors proposing PPO suggest a clipped objective function [149]

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(P_r(\theta)\hat{A}_t, \text{clip}(P_r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right], \tag{2.37}$$

where ϵ is a hyperparameter and commonly set to 0.2 as suggested in [149], and $P_r(\theta)$ denotes a probability ratio, measuring the difference between the new policy π_θ and the previous policy $\pi_{\theta_{old}}$, such that

$$P_r(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, \tag{2.38}$$

and thus $P_r(\theta_{old}) = 1.0$. The loss $\mathcal{L}^{CLIP}(\theta)$ consists of two terms. The first term is simply the objective function used in the Trust Region Policy Optimization (TRPO) algorithm, based on which PPO is built [149]. The second term is what makes PPO stable: $\text{clip}(P_r(\theta), 1-\epsilon, 1+\epsilon)$. This clipping function restricts updates within the interval $[1-\epsilon, 1+\epsilon]$, thus allowing updates only within a trust region. During the optimization, the smaller term of both is taken, such that a pessimistic lower bound of the objective is considered. This penalizes large changes in the gradient estimator and thus in the changes of the policy. In the original paper, the authors further suggest an optional adaptive Kullback Leibler (KL) divergence penalty term, which was not implemented for the PPO algorithm within this work, since the clipping of the objective function has shown to sufficiently stabilize the training.

Since PPO belongs to the actor-critic algorithms, two networks are employed, an actor network for the policy, and a critic network for the value function. The actor network models the policy and is trained to maximize the objective given in (2.37). The critic network with parameters ψ approximates the value function given in (2.23) as $V(\psi, s)$, and the loss is simply computed as the mean squared error between predicted values and the discounted reward-to-go $\hat{r}$ based on a general advantage estimation $\hat{A}$ [148]. Thus, the actor learns the policy, while the critic evaluates the taken actions by estimating the value of the encountered states. Using actor-critic frameworks in RL has shown to increase robustness of the training, with the value estimations of the critic network guiding the policy updates of the actor. The training algorithm of PPO is summarized in algorithm 3. The implementation of the actor and critic networks often involves shared layers, leading to a shared objective function by combining the loss terms of both actor and critic. For the considered problem of this thesis, the architectures of both networks and their loss terms are separated. All policies in this work follow a normal distribution during training, for which the standard deviation is set to a fixed value such that the actor network predicts the mean of the distribution. During predictions after the training, the actions are chosen deterministically by setting the standard deviation to zero. Other implementations exist, letting the actor predict both mean and standard deviation, which has shown to be less robust for this work. The greater the standard deviation during training is,

Algorithm 3 PPO Training

```
Initialize policy parameters $\theta_0$ and value function parameters $\psi_0$
for episode $e$ in $E$ or until stopping criterion is reached do
    Collect set of trajectories $D_e = \{\tau_i\}$ with policy $\pi(\theta_e)$ in environment
    Compute rewards-to-go $\hat{r}_t$
    Compute advantage estimates $\hat{A}_t$ based on value function $V(\psi_e)$
    Via an optimization algorithm update policy parameters $\theta_e$ by maximizing
    PPO objective:
        $\theta_{e+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_e|T} \sum_{\tau \in \mathcal{D}_e} \sum_{t=0}^{T} \min\left(\hat{P}_t(\theta_e)\hat{A}_t(\theta_e), clip(\epsilon, \hat{A}_t(\theta_e))\right)$
    Via some optimization algorithm update value function parameters $\psi_e$ by
    minimizing value loss function:
        $\phi_{e+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_e|T} \sum_{\tau \in \mathcal{D}_e} \sum_{t=0}^{T} (V_t(\psi_e) - \hat{r}_t)^2$
end for
Return actor
```

the more exploration is encouraged. Another approach to encourage exploration, implemented in the PPO algorithm in this work, is the addition of an entropy bonus to the actor's loss function [149, 171].

2.2.3 Generalization: Why Can Trained Models Be Used On New Data?

The central goal of ML is to train models that not only perform well on the data they are trained on but also generalize effectively to new data outside of the training dataset. This generalization capability renders ML models useful for real-world applications. Understanding key concepts such as data distribution, and over- and underfitting is essential for achieving robust model performance, ultimately ensuring that the models do not simply memorize the training data but are adaptable to new examples.

Independent and Identically Distributed versus Out-of-Distribution

ML methods are inherently statistical techniques, and are thus heavily affected by data distribution. A fundamental concept of ML is whether the data is independent and identically distributed (i.i.d.) or out-of-distribution (o.o.d.) [18]. i.i.d. refers to a dataset containing observations which are drawn independently of each other but from the same underlying probability distribution. Employing an ML model, it is often assumed that both train and test data are i.i.d. [18], as given on the left hand side of Figure 2.5. With this assumption, the model has a clear and consistent relationship to learn, and its performance on new, unseen data following the same distribution is likely to be reliable. o.o.d. describes data significantly differing from the training data. This could originate from a shift in the distribution of the model inputs or outputs, or due to new data which is sampled from a different domain than the training data, as depicted on the right of Figure 2.5. A trained model which only reflects the patterns of the training data distribution will show a significant degradation in performance when employed on o.o.d. data, as this requires extrapolation capabilities. This scenario is a common problem in real-world applications, such as for example in time series [18]. Data from a specific time period may only capture current conditions, meaning that future data could differ significantly due to for example seasonal changes. This results in o.o.d. data that doesn't align with past trends, rendering long-term predictions inherently difficult.

In this work, to generate data samples within a given design space, stochastic sampling methods are primarily used, employing the SMARTy library [13]. This, since equidistant or full factorial sampling, depicted in Figure 2.6a, may miss fine-scale variations if they align poorly with the function being learned or may over-represent regular structures, potentially biasing the model. Choosing random or quasi-random distributions, for example with a Halton sampling strategy given in Figure 2.6b, helps the model generalize better, as it encounters more diverse variations in the design space rather than regularized patterns.

Model Complexity: Bias vs. Variance

Beside the data distribution, the proper selection of the ML model and the tuning of its hyperparameters have a significant effect on the final performance. Balancing model complexity is key to avoid under- and overfitting, concepts depicted in Figure 2.7. Underfitting occurs when a trained model is too simplistic, often exhibiting a high bias, to capture the relation between in- and outputs, resulting in poor performance even for the training dataset, as for example a linear model for non-linear problems. Overfitting occurs if the model captures not only the underlying patterns but also noise in the training data, a problem often encountered for models with high variance. This leads to excellent performance on the training set, but the patterns learned do not generalize well to unseen test data, resulting in poor performance outside of the training environment. Thus, careful model selection, finding the right hyperparameters, as well as monitoring and comparing model performance during training on a separate validation dataset is essential

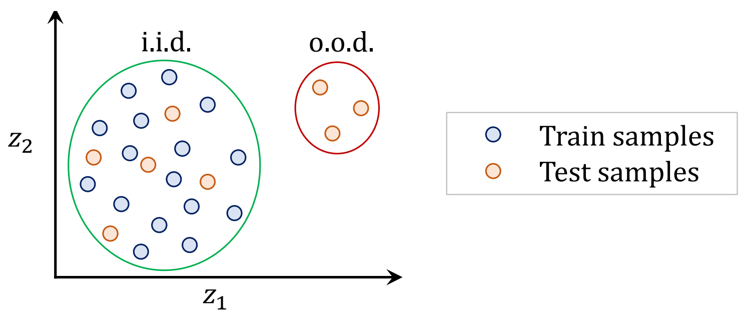
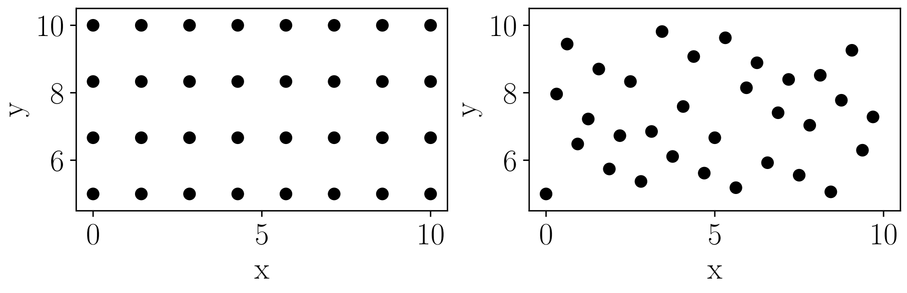


Figure 2.5: i.i.d. vs. o.o.d. data



(a) Full factorial sampling (b) Halton sampling

Figure 2.6: Data sampling strategies

for detecting and preventing these effects.

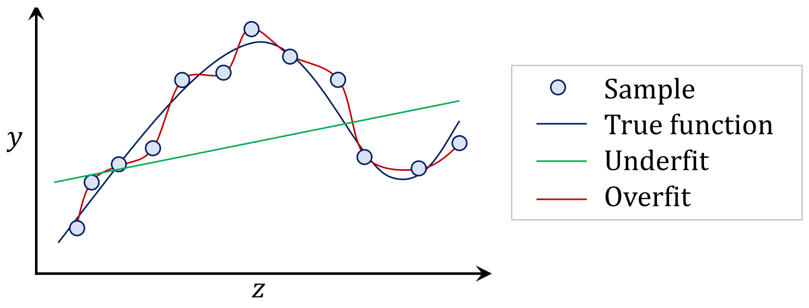


Figure 2.7: Underfit vs. overfit

2.2.4 Model Selection and Performance Evaluation

To build robust and reliable models, various tools and strategies address common challenges. Model performance is assessed through validation metrics to ensure generalization across data. A standard ML practice is to split the dataset into training, validation, and test sets: the model is trained on the first, tuned using the second, and finally evaluated on the third.

The following sections cover validation metrics and hyperparameter optimization strategies. Given this work's focus on external aerodynamics, the trained ML model performance is evaluated not only using standard ML metrics but also through aerodynamic coefficients, which will be defined in the following.

Validation Metrics and Strategies

To assess a model's performance, clear criteria defining a successful model are needed. For this, various validation metrics exist to judge the quality of a model. Some metrics are applicable only to either regression or classification tasks. Furthermore, any metric comes with specific advantages and disadvantages, thus it is necessary to assess and compare models not only by a single one. For the present work focusing on regression tasks, three metrics are chosen. The first validation metric introduced is the Mean Squared Error (MSE), defined as

$$\mathrm{MSE} = \frac{1}{J}\sum_{j=1}^{J}(y_j - \hat{y}_j)^2, \tag{2.39}$$

where J is the number of instances or observations, y_j the ground truth value, and $\hat{y}_j$ the predicted value. Thus, the MSE squares the differences between predicted and true value, penalizing larger errors. This characteristic might pose a disadvantage, since outliers forming greater errors have more influence on the overall MSE score.

The second metric introduced is the Mean Absolute Error (MAE), which is unlike the MSE less sensitive to outliers, and is given as

$$\mathrm{MAE} = \frac{1}{J}\sum_{j=1}^{J}|y_j - \hat{y}_j|. \tag{2.40}$$

Thus, all errors are treated linearly and larger errors do not receive greater attention. The final validation metric being considered in this work is the coefficient of determination, also known as R^2, which is defined as

$$R^2 = 1 - \frac{\sum_{j=1}^{J}(y_j - \hat{y}_j)^2}{\sum_{j=1}^{J}(y_j - \bar{y})^2}, \tag{2.41}$$

where $\bar{y}$ denotes the mean of the ground truth values. A model predicting the exact output such that $\hat{\boldsymbol{y}} = \boldsymbol{y}$ will reach a coefficient of determination $R^2 = 1$, whereas $R^2 = 0$ corresponds to a model predicting the average $\bar{y}$ for any given instance. The R^2 assumes a linear relationship between in- and outputs, making it reliable for linear problems. For non-linear relationships, relying solely on this metric might be misleading and might indicate an over- or underfit, even if this is not the case. Nevertheless, an advantage of this metric is its intuitive interpretation with a normalized scale up to $R^2 = 1$, which is a helpful assessment when used together with other quantitative metrics, even for non-linear problems [38].

These proposed metrics can be used as part of a validation strategy to find optimal hyperparameters or to compare different models with each other. In this work, two strategies for splitting the dataset for validation are considered, namely hold-out validation and k-fold validation. For hold-out validation, the total available data is split into a training set and smaller validation and test sets, which do not change during the training. K-fold validation is more intricate and requires k training cycles for the final validation score. With this strategy, the validation data consists of a different portion of the entire training dataset for each fold. These cycles can become time consuming and computational expensive. Nevertheless, k-fold validation uses the available data more efficiently by using each data instance for both training and validation, and helps reducing the risk of overfitting.

Aerodynamic Coefficients

The previously presented metrics are valuable for hyperparameter tuning using the validation set and for evaluating the model's performance on the test set. Nevertheless, this work is mainly concerned with external aerodynamic problems. Thus, it is crucial to assess how well the final ML model corrects and improves key aerodynamic values on the test samples.

The first dimensionless quantity of interest is the pressure coefficient C_P, describing the relative pressure at a surface point compared to free-stream conditions:

$$C_P = \frac{P - P_\infty}{\frac{1}{2}\rho_\infty U_\infty^2}, \tag{2.42}$$

with P_∞ the free-stream pressure, ρ_∞ the free-stream air density, and U_∞ the free-stream velocity.

Secondly, the lift coefficient C_L describing the generated lift force F_L relative to the dynamic pressure $\frac{1}{2}\rho_\infty U_\infty^2$ and reference area A is another dimensionless parameter of interest and given as:

$$C_L = \frac{F_L}{\frac{1}{2}\rho_\infty U_\infty^2 A}. \tag{2.43}$$

Similarly, the drag coefficient C_D represents the ratio between the drag force F_D acting on a body and the dynamic pressure and reference area:

$$C_D = \frac{F_D}{\frac{1}{2}\rho_\infty U_\infty^2 A}. \tag{2.44}$$

Finally, the last dimensionless value of interest is the skin friction coefficient C_f. It quantifies the wall shear stress τ_w at a surface point with respect to the dynamic pressure:

$$C_f = \frac{\tau_w}{\frac{1}{2}\rho_\infty U_\infty^2}. \tag{2.45}$$

The wall shear stress τ_w is defined by the dynamic viscosity μ_{dyn}, the velocity component v parallel to the surface, and y, the normal distance to the surface:

$$\tau_w = \mu_{\text{dyn}}\left(\frac{\partial v}{\partial y}\right)_{y=0}. \tag{2.46}$$

Hyperparameter Optimization

Even if ML models do not need to be explicitly programmed to learn patterns from data, their hyperparameters need to be explicitly set prior to the training. This might pose one of the most cumbersome tasks in the field of deep learning [2], if one considers how many hyperparameters a NN entails and how sensitive the results are with respect to their choice. Thus, a great research effort focuses on algorithms for HyperParameter Optimization (HPO).

The crudest option to do HPO belonging to the category of model-free algorithms is simple trial and error - also called babysitting or grad student descent (GSD) because of its popularity among students [175] - which consists of finding hyperparameters manually. Obviously, this approach should be abandoned early for complex problems with many hyperparameters. Two other model-free algorithms include grid search and random search. Although both are easily parallelized, they are not efficient. Assume a grid search with k hyperparameters, each of which having n values, the computational complexity for grid search is $\mathcal{O}(n^k)$ [116]. Random search slightly reduces the computational effort by only sampling a fixed number of hyperparameter combinations in the grid. Nevertheless, since both algorithms do not exploit regions in the search space which have previously performed well, they will conduct unnecessary training runs.

Gradient-based optimization forms another category of possible algorithms. Although effective, since gradients allow to identify and move along the direction towards the optimum, they are rarely employed for HPO [175]. The main reason is that they can only be used for continuous hyperparameters for which gradients can be computed, like the learning rate for NNs, but not for discrete or categorical ones.

Yet another category is Bayesian optimization, a popular choice for HPO [175]. Bayesian optimization takes into account previously encountered hyperparameter combinations to determine the next configuration, trying to find a balance between exploration of new promising regions and exploitation of currently promising areas. A surrogate model and an objective function form the two main ingredients for any Bayesian optimization algorithm [77]. The surrogate model is fit on previously encountered samples and the objective function values. The next hyperparameter combination is selected based on a selected acquisition function or infill criterion [154]. This combination is then used to evaluate the real objective function and the surrogate model is updated. This procedure is repeated until a stopping criterion is reached. Since the procedure is sequential, these algorithms are difficult to parallelize. Nevertheless, this approach has shown to be more efficient than model-free algorithms, often detecting sufficiently good hyperparameters after a few iterations [175]. Most commonly used surrogate models for HPO are Gaussian processes, random forests, and Tree Parzen Estimators (TPE) [175]. In this work, for more intricate model training, Bayesian optimization with TPE is employed, if not stated differently, using the Optuna library [2].

Chapter 3

Methodology

The following chapter presents different correction methodologies applied in this thesis. Section 3.1 defines the correction terms for FV and DG methods, applicable to steady and unsteady simulations. For FV, the variables of interest are directly corrected, whereas for DG, these variables are indirectly adjusted by correcting the polynomial coefficients. The aim is to correct the entire flow field, while allowing flexibility through a point by point correction, i.e. vertex by vertex for a vertex-centered FV scheme and element by element for the DG scheme. This allows to extract a significant amount of training instances per simulation. Additionally, correcting the flow field allows firstly to extract not only certain values such as loads, but also 3D flow phenomena such as vortices, and secondly enables the predicted correction term to interact with the CFD solver across the entire flow regime if necessary. In the following, the corrections, whether for the FV or DG discretization, are applied either as post-processing for steady simulations, or in-between CFD solver iterations for unsteady simulations.

Section 3.2.1 and 3.2.2 present the post-processing correction methodology for FV and DG solutions, respectively. Both consist of a supervised learning approach of three steps: data generation, training, and prediction for scenarios not present in the training data. While the post-processing correction of FV solutions is similarly extendable to unsteady simulations, the final section 3.3 focuses on the correction of low-fidelity unsteady DG simulations. For the unsteady correction, different correction approaches in combination with different training methods are presented: a post-processing correction with supervised learning, an iterative correction applied in-between CFD solver iterations trained under supervised learning, and an iterative correction employing a training based on RL.

3.1 Definition of the Correction Term

Let $\boldsymbol{u}(x,t)$ be the variable of interest to be solved for and $\tilde{\boldsymbol{u}}(x,t)$ its approximated solution at time t. This work considers the residual $R(\tilde{\boldsymbol{u}}) = 0$ [98, 165]. R

corresponds to a boundary value problem of the equations of interest as presented in 2.1.1.

3.1.1 Finite Volume Correction

The variables of interest on the coarse and fine grid are $\tilde{\boldsymbol{u}}_c$ and $\tilde{\boldsymbol{u}}_f$, where the subscripts c and f denote coarse and fine grid related variables, respectively. For the FV study of this work, the aim is to quantify and learn the discretization error on the coarse grid. Thus, the fine grid solution $\tilde{\boldsymbol{u}}_f$ needs to be projected to the coarse grid discretization by introducing a fine-to-coarse mapping operator $\mathcal{I}_f^c$:

$$\tilde{\boldsymbol{u}}_{I,c} = \mathcal{I}_f^c(\tilde{\boldsymbol{u}}_f). \tag{3.1}$$

Subsequently, the error $\Delta\tilde{\boldsymbol{u}}_c$ between the coarse grid solution $\tilde{\boldsymbol{u}}_c$ and the projected solution $\tilde{\boldsymbol{u}}_{I,c}$ can be quantified as:

$$\Delta\tilde{\boldsymbol{u}}_c = \tilde{\boldsymbol{u}}_{I,c} - \tilde{\boldsymbol{u}}_c. \tag{3.2}$$

The ML model receives inputs derived from the coarse grid solution, denoted as $\boldsymbol{\eta}(\tilde{\boldsymbol{u}}_c)$. The aim of the ML training is to find a relationship between the feature vector $\boldsymbol{\eta}(\tilde{\boldsymbol{u}}_{\boldsymbol{c}})$ and the target variable $\Delta\tilde{\boldsymbol{u}}_c$:

$$f_{ML}(\boldsymbol{\eta}) = \Delta\hat{\boldsymbol{u}}_c, \tag{3.3}$$

$$\Delta\tilde{\boldsymbol{u}}_c = f_{ML}(\boldsymbol{\eta}) + \varepsilon, \tag{3.4}$$

where ε is the deviation between the model's prediction $\Delta\hat{\boldsymbol{u}}_c$ and the true target value $\Delta\tilde{\boldsymbol{u}}_c$. To reduce this error, a loss function $\mathcal{L}(\Delta\tilde{\boldsymbol{u}}_c, \Delta\hat{\boldsymbol{u}}_c)$ measuring the discrepancy between predictions and true values is minimized by optimizing the model's parameters θ during training.

After the training, the prediction is used to correct coarse grid simulations unseen during the training, resulting in the corrected unknown $\hat{\boldsymbol{u}}_c$:

$$\hat{\boldsymbol{u}}_c = \tilde{\boldsymbol{u}}_c + \Delta\hat{\boldsymbol{u}}_c. \tag{3.5}$$

The corrected solution approximates the projected fine grid solution on the coarse grid discretization $\tilde{\boldsymbol{u}}_{I,c}$, such that

$$\hat{\boldsymbol{u}}_c \approx \tilde{\boldsymbol{u}}_{I,c}. \tag{3.6}$$

Injection as Mapping Operator

For the cases considered in this work, injection is used as a mapping operator I_f^c. With $I_f^c : V_f \to V_c$, where V_f and V_c are the function spaces of fine and coarse grid Ω_f and Ω_c, respectively, the injection is defined as

$$(I_f^c\tilde{\boldsymbol{u}}_f)(x_i^c) = \tilde{\boldsymbol{u}}_f(x_i^c) \tag{3.7}$$

for all $x_i^c \in \Omega_c$ being the locations of the coarse grid nodes. Since a series of nested grids is used, it follows that $\Omega_c \subset \Omega_f$. Injection as mapping operator between two nested grids is exemplarily presented in Figure 3.1

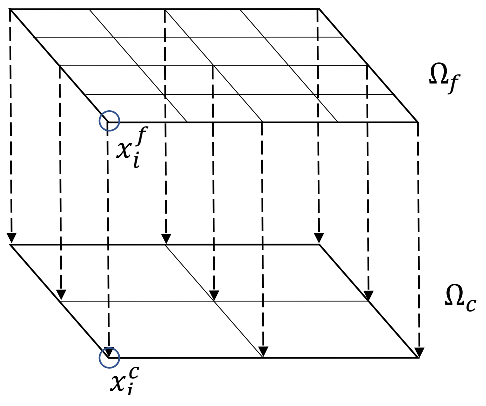


Figure 3.1: Injection as mapping operator I_f^c on nested grids $\Omega_c \subset \Omega_f$

Injection preserves the exact values at coarse grid nodes when these nodes coincide with fine grid vertices, ensuring an accurate representation of quantities such as the pressure coefficient along an airfoil surface. This property holds for vertex-centered FV codes and nested grids. Nevertheless, injection also leads to a loss of data from the high-fidelity discretization, as all other fine grid points are discarded, potentially neglecting fine grid variations in the solution. If surface related values, such as the pressure coefficient, are not necessarily in focus, or if non-nested grids are used, alternative methods such as volume weighted interpolations may be more suitable. Such methods are often used in the multigrid method [65, 160] to capture volume averaged quantities and minimize interpolation artifacts across the flow field.

3.1.2 Discontinuous Galerkin Correction

Employing a DG discretization with orthonomal basis function, the numerical solution $\tilde{\boldsymbol{u}}$ at time t is expressed in each element K as piece-wise polynomial function, as given in (2.14), with $\boldsymbol{a}$ being the polynomial coefficients, $\boldsymbol{\phi}$ the basis functions, and N the number of degrees of freedom per equation per element. Let the low-order and high-order solutions be denoted by subscripts LO and HO, respectively. Thus, the variables of interest become $\tilde{\boldsymbol{u}}_{\mathrm{LO}}$ and $\tilde{\boldsymbol{u}}_{\mathrm{HO}}$ defined by $\boldsymbol{a}_{\mathrm{LO}}$ and $\boldsymbol{a}_{\mathrm{HO}}$, and it is assumed that $N_{\mathrm{LO}} < N_{\mathrm{HO}}$.

As for the FV correction, the discretization error is defined on the low-order discretization and later to be learned by the ML model. Thus, the high-order coefficients are truncated to the low-order discretization, resulting in the truncated solution $\tilde{\boldsymbol{u}}_{T,\mathrm{LO}}$:

$$\tilde{\boldsymbol{u}}_{T,\mathrm{LO}} = \sum_{i=1}^{N_{\mathrm{LO}}} a_{i,\mathrm{HO}} \boldsymbol{\phi}_i. \tag{3.8}$$

Since the basis is hierarchical and orthonormal, this results in an L_2 projection. The error is computed as the difference between the truncated and low-order coefficients:

$$\Delta a_i = a_{i,\mathrm{HO}} - a_{i,\mathrm{LO}}, \tag{3.9}$$

for $1 \leq i \leq N_{\mathrm{LO}}$. As before, a loss function $\mathcal{L}(\Delta \boldsymbol{a}, \Delta \hat{\boldsymbol{a}})$ is minimized during ML training to find a relationship between the feature vector $\boldsymbol{\eta}(\hat{\boldsymbol{u}}_{\boldsymbol{LO}})$ and the target variable $\Delta \boldsymbol{a}$:

$$f_{ML}(\boldsymbol{\eta}) = \Delta \hat{\boldsymbol{a}}, \tag{3.10}$$

$$\Delta \boldsymbol{a} = f_{ML}(\boldsymbol{\eta}) + \varepsilon, \tag{3.11}$$

with ε the difference between the model's prediction $\Delta \hat{\boldsymbol{a}}$ and the true target value $\Delta \boldsymbol{a}$. With the ML predictions $\Delta \hat{\boldsymbol{a}}$, the LO solution $\tilde{\boldsymbol{u}}_{\mathrm{LO}}$ is corrected as

$$\hat{\boldsymbol{u}}_{\mathrm{LO}} = \sum_{i=1}^{N_{\mathrm{LO}}} (a_{i,\mathrm{LO}} + \Delta \hat{a}_i) \boldsymbol{\phi}_i. \tag{3.12}$$

The corrected solution $\hat{\boldsymbol{u}}_{\mathrm{LO}}$ approximates the truncated one $\tilde{\boldsymbol{u}}_{T,\mathrm{LO}}$:

$$\hat{\boldsymbol{u}}_{\mathrm{LO}} \approx \tilde{\boldsymbol{u}}_{T,\mathrm{LO}}. \tag{3.13}$$

Finally, a connection between corrections of FV and DG discretizations can be established: for polynomial degree $p = 0$ with $\phi_1 = 1$, the DG correction equation (3.12) corresponds to the FV correction given in (3.5).

3.2 Post-Processing Correction of Steady Simulations

In the following sections, the workflow for correcting steady simulations is described. First, it is presented for an FV discretization, followed by a DG scheme.

3.2.1 Coarse Grid Finite Volume Correction

This section describes the correction approach applied to steady coarse grid FV simulations. The aim is to find a ML based correction function to increase the accuracy of coarse grid simulations. The corrected solution ultimately approximates the fine grid solution projected onto the coarse grid discretization. Figure 3.2 illustrates the proposed supervised learning approach, consisting of three steps: dataset generation, ML training, and ML model prediction.

In the first step, as depicted on the left in Figure 3.2, the dataset for the training is generated. This is done by running the same simulations on both coarse and fine grid discretizations. The simulations are conducted on different sample points within the design space. In the cases considered, the design space is spanned by Mach number M and angle of attack α. Differences in the coarse and fine solutions $\tilde{\boldsymbol{u}}_c$ and $\tilde{\boldsymbol{u}}_f$ arise due to the coarse grid induced error. After this, the fine solutions are projected onto the coarse discretization using a mapping operator $\mathcal{I}_f^c$, as previously described in (3.1). Subsequently, the correction term

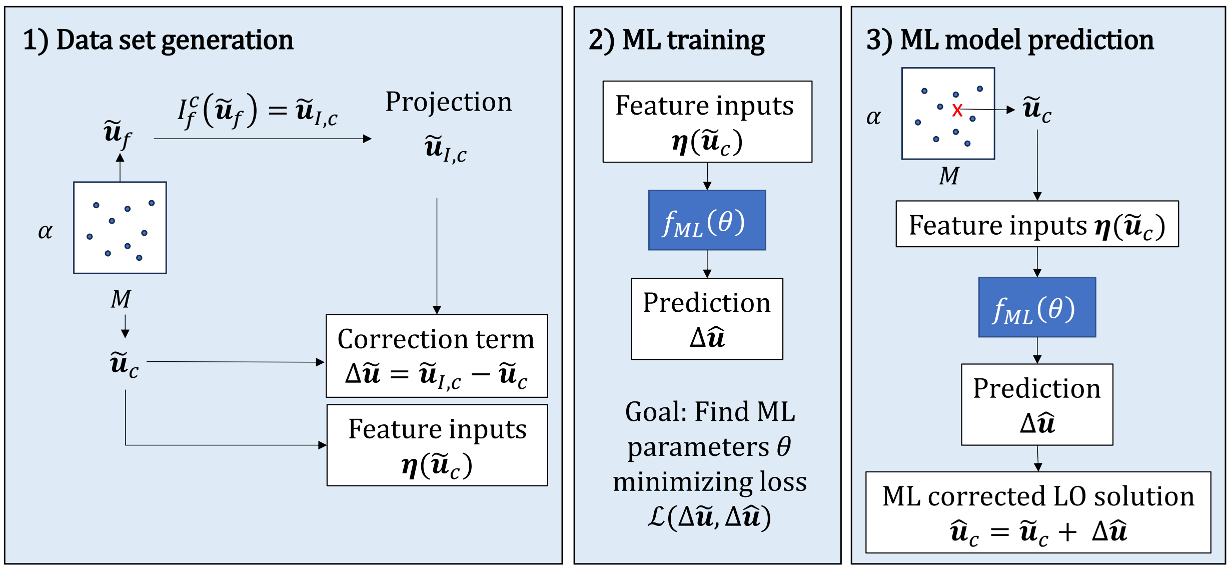

1) Data set generation
$I_f^c(\tilde{\boldsymbol{u}}_f) = \tilde{\boldsymbol{u}}_{I,c}$
Projection
$\tilde{\boldsymbol{u}}_f$
$\tilde{\boldsymbol{u}}_{I,c}$
α
M
$\tilde{\boldsymbol{u}}_c$
Correction term
$\Delta\tilde{\boldsymbol{u}} = \tilde{\boldsymbol{u}}_{I,c} - \tilde{\boldsymbol{u}}_c$
Feature inputs
$\boldsymbol{\eta}(\tilde{\boldsymbol{u}}_c)$
2) ML training
Feature inputs
$\boldsymbol{\eta}(\tilde{\boldsymbol{u}}_c)$
$f_{ML}(\theta)$
Prediction
$\Delta\hat{\boldsymbol{u}}$
Goal: Find ML parameters θ minimizing loss $\mathcal{L}(\Delta\tilde{\boldsymbol{u}}, \Delta\hat{\boldsymbol{u}})$
3) ML model prediction
α
M
$\tilde{\boldsymbol{u}}_c$
Feature inputs $\boldsymbol{\eta}(\tilde{\boldsymbol{u}}_c)$
$f_{ML}(\theta)$
Prediction
$\Delta\hat{\boldsymbol{u}}$
ML corrected LO solution
$\hat{\boldsymbol{u}}_c = \tilde{\boldsymbol{u}}_c + \Delta\hat{\boldsymbol{u}}$

Figure 3.2: Overview of steady FV correction workflow

is computed as the difference between coarse grid solution and projected solution at each point of the grid.

Additionally, several quantities are derived from the low-fidelity solution, making up the feature vector $\boldsymbol{\eta}(\tilde{\boldsymbol{u}}_c)$. As a first feature, a local cell Reynolds number Re_k is chosen, which is computed using the local velocity magnitude U_k, wall distance d_k, and molecular viscosity ν_k, and is given as

$$Re_k = \frac{|U_k|d_k}{\nu_k} \tag{3.14}$$

Furthermore, it can be assumed that the discretization error correlates to steep gradients within a flow solution. Thus, the first and second derivatives of the variables of interest of the coarse grid solution are a natural choice as additional features. The derivatives are computed using a weighted least squares approach with an inverse distance weighting [169]. Thus, the feature vector for each vertex k can be written as

$$\boldsymbol{\eta}(\tilde{\boldsymbol{u}}_c) = \left[\frac{\partial \tilde{\boldsymbol{u}}_c}{\partial x_i}\bigg|_k, \frac{\partial^2 \tilde{\boldsymbol{u}}_c}{\partial x_i \partial x_j}\bigg|_k, Re_k \right]. \tag{3.15}$$

This set of features serves as input of the ML model, while the correction term is the target for the machine learning model during the second step, the ML training. Here, the model is trained by adjusting its parameters θ, minimizing the loss function $L(\Delta\tilde{\boldsymbol{u}}_c, \Delta\hat{\boldsymbol{u}}_c)$. Once trained, the parameters are frozen, and the ML model can be employed to predict the correction term for any kind of coarse grid simulations, which have not been encountered during the training phase, as illustrated on the right of Figure 3.2 by a red cross in the design space. The feature inputs have to be the same, thus local Reynolds number, and the first and second derivative of the coarse grid solution have to be computed prior to any prediction.

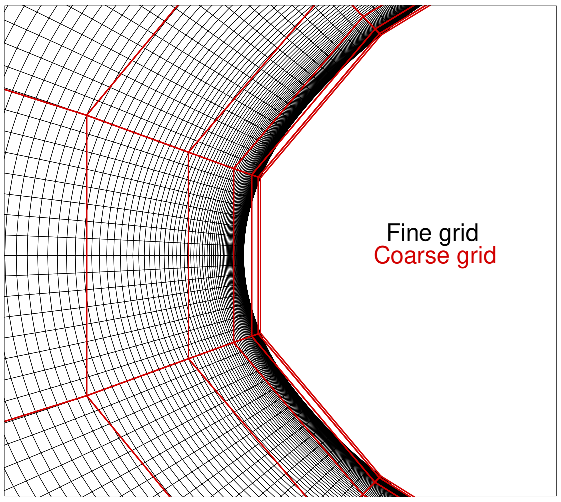


Figure 3.3: Nested grids: coarse vs. fine around curved boundary

The corrected flow field is obtained by adding the prediction to the coarse grid variables of interest, as described in (3.5).

In chapter 4, the proposed method is demonstrated on a 2D and a 3D test case, including the RAE2822 airfoil and the LANN wing geometry. The corrected flow fields are analyzed, and derived values, such as pressure coefficients on the surface and lift coefficient, are assessed. This allows to showcase strengths and limitations of an ML based correction for coarse grid steady FV simulations.

Choice of the mapping operator and coarse grid.

Using a mapping operation $\mathcal{I}_f^c$ to project the fine grid data onto the coarse grid discretization poses the following limitations to the proposed procedure:

1. The choice of $\mathcal{I}_f^c$ is non-trivial, especially around curvatures if the data points of the fine and the coarse grid are not overlapping. This is for example the case for cell-centered data and nested grids, where the coarse boundary layer cells are located outside of the fine grid geometry, as illustrated in Figure 3.3.

2. The ML corrected coarse grid solution can only be expected to approximate the mapped solution, since it is trained on the projected and not the fine grid data. By reducing the number of data points, the mapping operation from fine to coarse naturally decreases the high-fidelity information content. Thus, the maximum accuracy of the corrected solution is limited by two factors: one, the coarse grid discretization itself, and two, by the chosen mapping function $\mathcal{I}_f^c$.

Point-wise training and prediction.

Choosing a correction acting on each data point separately, i.e. on the data in each grid vertex or element, allows for greater flexibility. Training on the whole flow domain would limit the model to be applicable only to the specific grid seen during training. Another advantage is that a point-wise correction approach results in more training instances per simulation and reduces the dimensionality of the in- and output. The disadvantage of this approach is that the relation between in- and output might be non-unique: two different points in one solution field might have the same value for the unknown $\tilde{\boldsymbol{u}}_c$ but might need two different corrections. This poses a problem for any training approach taking only local information into account. Clever selection of features can minimize this issue: in this work, the local cell Reynolds number Re_k includes the wall distance d_k and the computation of the derivatives takes into account information from the neighborhood. Another measure to avoid this issue is deploying a model which accumulates neighboring information for a point-wise prediction, such as CNNs or GNNs.

3.2.2 Low-order Discontinuous Galerkin Correction

The post-processing correction of low-fidelity steady DG simulations obtained with low polynomial degree follows a similar approach to the FV correction described in the previous section. The supervised learning approach is depicted in Figure 3.4, consisting again of three steps, namely dataset generation within a design space spanned by Mach number M and angle of attack α, ML training, and ML model prediction.

One of the main differences to the FV correction is that both low- and high-fidelity simulations can be conducted on the same grid to generate the dataset. Different solutions are obtained by using varying polynomial degrees, such that $N_{LO} \leq N_{HO}$. With a hierarchical and orthonormal basis, an L_2 projection of the high-order solution is achieved, by simply truncating the higher-order terms to the low-order discretization. The error or correction term is then computed as the difference between the truncated coefficients $\boldsymbol{a}_T$ and the low-order coefficients $\boldsymbol{a}_{LO}$. For the feature inputs, the cell Reynolds number is computed as given

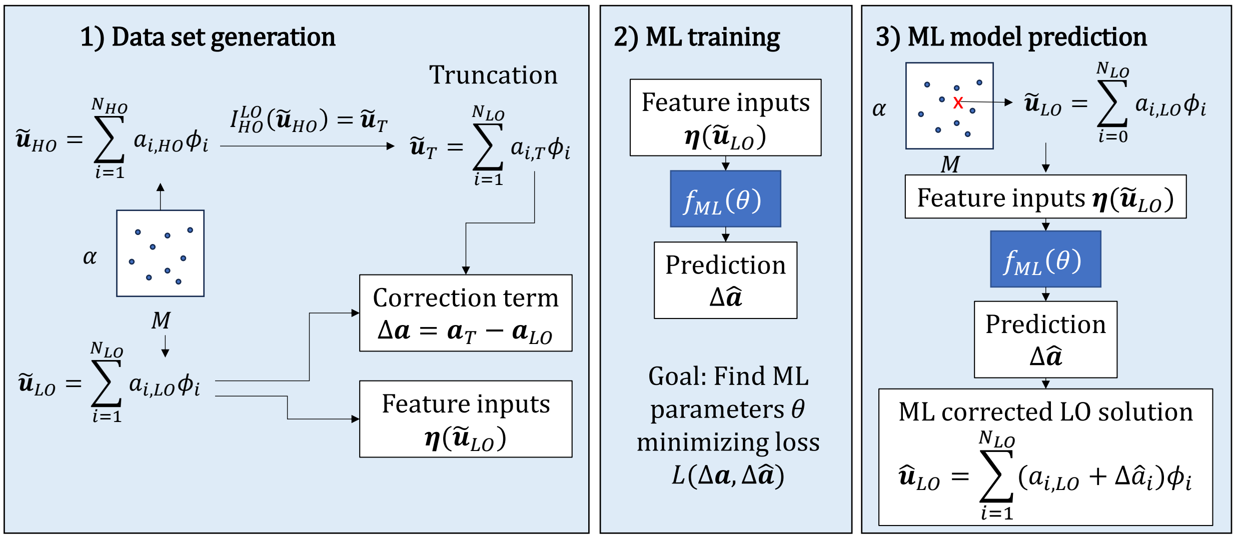

1) Data set generation
Truncation
$\tilde{\boldsymbol{u}}_{HO} = \sum_{i=1}^{N_{HO}} a_{i,HO}\phi_i$
$I_{HO}^{LO}(\tilde{\boldsymbol{u}}_{HO}) = \tilde{\boldsymbol{u}}_T$
$\tilde{\boldsymbol{u}}_T = \sum_{i=1}^{N_{LO}} a_{i,T}\phi_i$
α
M
$\tilde{\boldsymbol{u}}_{LO} = \sum_{i=1}^{N_{LO}} a_{i,LO}\phi_i$
Correction term $\Delta\boldsymbol{a} = \boldsymbol{a}_T - \boldsymbol{a}_{LO}$
Feature inputs $\boldsymbol{\eta}(\tilde{\boldsymbol{u}}_{LO})$
2) ML training
Feature inputs $\boldsymbol{\eta}(\tilde{\boldsymbol{u}}_{LO})$
$f_{ML}(\theta)$
Prediction $\Delta\hat{\boldsymbol{a}}$
Goal: Find ML parameters θ minimizing loss $L(\Delta\boldsymbol{a}, \Delta\hat{\boldsymbol{a}})$
3) ML model prediction
α
M
$\tilde{\boldsymbol{u}}_{LO} = \sum_{i=0}^{N_{LO}} a_{i,LO}\phi_i$
Feature inputs $\boldsymbol{\eta}(\tilde{\boldsymbol{u}}_{LO})$
$f_{ML}(\theta)$
Prediction $\Delta\hat{\boldsymbol{a}}$
ML corrected LO solution
$\hat{\boldsymbol{u}}_{LO} = \sum_{i=1}^{N_{LO}} (a_{i,LO} + \Delta\hat{a}_i)\phi_i$

Figure 3.4: Overview of steady DG correction workflow

in (3.14). The first and second derivatives are computed with respect to the polynomial coefficients. Thus, the feature vector for an element k can in general be written as

$$\boldsymbol{\eta}(\tilde{\boldsymbol{u}}_{\boldsymbol{LO}}) = \left[\frac{\partial \boldsymbol{a}_{LO}}{\partial x_i}\bigg|_k, \frac{\partial^2 \boldsymbol{a}_{LO}}{\partial x_i \partial x_j}\bigg|_k, Re_k \right]. \tag{3.16}$$

Consequently, as the polynomial degree of the solution to be corrected increases, the number of feature inputs and correction outputs grows. The ML training and the prediction steps follow closely the previously described ones for the FV correction: the ML model is trained by providing the set of inputs and outputs, trying to find optimal model parameters θ by minimizing a loss function $L(\Delta\boldsymbol{a}, \Delta\hat{\boldsymbol{a}})$. Subsequently, the trained model is employed to make predictions on new low-fidelity simulations, as marked by the red cross in the design space, and the correction term can be used to approximate an improved flow field $\hat{\boldsymbol{u}}_{LO}$.

The DG post-processing method is tested on a 2D and 3D case and presented in chapter 5, including the RAE2822 airfoil and a delta wing geometry. As for the FV correction, assessments of the corrected flow field and several derived values are made. The results demonstrate advantages and limitations of using an ML based correction for DG solutions.

Choice of low-fidelity polynomial degree.

Notably, aerodynamic RANS simulations are commonly conducted only up to $p = 4$, often showing sufficiently accurate results with $p = 3$ [24, 28]. Thus, correcting RANS solutions obtained with polynomial degree $p \in \{0, 1, 2\}$ is a reasonable choice. Nevertheless, the applications considered in this work restrict the correction to $p \in \{0, 1\}$, as the problems considered show sufficient accuracy with polynomial degree $p = 2$. Additionally, the number of in- and outputs and therefore the training cost do not only rise significantly with increased polynomial degree, but also with the dimension of the problem, as given in (2.17) and described in table 2.1.

3.3 Correction of Unsteady Simulations

This section presents correction approaches for unsteady low-fidelity simulations. Figure 3.5 depicts various trajectories of the numerical solution $\tilde{\boldsymbol{u}}(x, t)$ over time. The high-fidelity solution is illustrated in blue, whereas the low-fidelity trajectory is colored black. The figure qualitatively highlights the need for smaller time steps Δt for high-fidelity discretizations. Furthermore, even with both high- and low-fidelity simulations starting with the same initial conditions, their trajectories diverge over time. Source of the inaccuracies of low-fidelity simulations are both spatial and temporal discretization errors. To correct the low-fidelity solution and approximate the respective high-fidelity solution, latter is truncated to the low-fidelity discretization, as shown by the green trajectory in Figure 3.5 and given in (3.8).

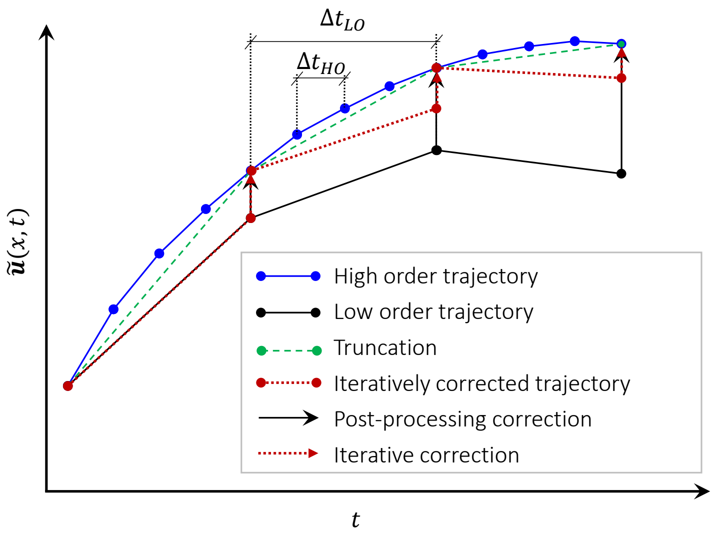


Figure 3.5: Different solution trajectories and correction methods

Table 3.1: Overview of correction and learning approaches

	Correction Method	**Learning Method**
1	Post-processing correction	Supervised learning
2	Iterative correction	Supervised learning
3	Iterative correction	Supervised learning + noise regularization
4	Iterative correction	Reinforcement learning

Two correction approaches are investigated, as depicted by the arrows in Figure 3.5. The first approach is referred to as *post-processing correction*, while the second one is denoted as *iterative correction*. The first method is an extension of the previously described post-processing correction for steady simulations to unsteady simulations, employing a supervised learning approach. Here, the correction is decoupled from the CFD solver and can be applied once the low-fidelity simulation has been concluded. For the other approach, the iterative correction and the CFD solver take turns, such that after each solver iteration the correction is being applied.

To learn the iterative correction, two different training procedures are employed: supervised learning and reinforcement learning. Additionally, a supervised learning approach regularized with noisy input features is investigated. This results in four different combinations of correction and training methods, as summarized in table 3.1. Similar to the previously described post-processing correction for steady simulations, the methods are applicable to both coarse grid FV and low polynomial degree DG solutions. The focus of this work will be the correction of latter. The methodology for each combination is described in the following sections.

In chapter 6, all approaches are first tested and compared on a simple 1D linear

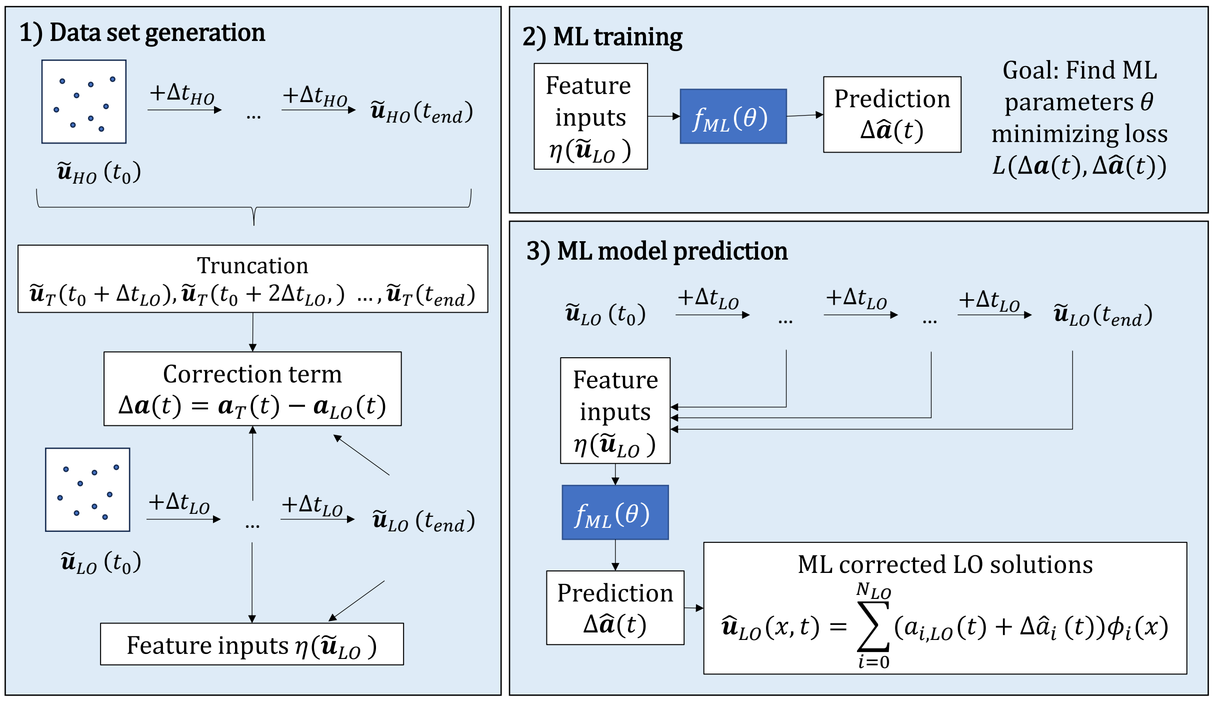

1) Data set generation
$+\Delta t_{HO}$
$+\Delta t_{HO}$
$\tilde{\boldsymbol{u}}_{HO}(t_{end})$
$\tilde{\boldsymbol{u}}_{HO}(t_0)$
Truncation
$\tilde{\boldsymbol{u}}_T(t_0+\Delta t_{LO}), \tilde{\boldsymbol{u}}_T(t_0+2\Delta t_{LO},) \dots, \tilde{\boldsymbol{u}}_T(t_{end})$
Correction term
$\Delta \boldsymbol{a}(t) = \boldsymbol{a}_T(t) - \boldsymbol{a}_{LO}(t)$
$+\Delta t_{LO}$
$+\Delta t_{LO}$
$\tilde{\boldsymbol{u}}_{LO}(t_{end})$
$\tilde{\boldsymbol{u}}_{LO}(t_0)$
Feature inputs $\eta(\tilde{\boldsymbol{u}}_{LO})$
2) ML training
Feature inputs $\eta(\tilde{\boldsymbol{u}}_{LO})$
$f_{ML}(\theta)$
Prediction $\Delta\hat{\boldsymbol{a}}(t)$
Goal: Find ML parameters θ minimizing loss $L(\Delta \boldsymbol{a}(t), \Delta\hat{\boldsymbol{a}}(t))$
3) ML model prediction
$\tilde{\boldsymbol{u}}_{LO}(t_0)$
$+\Delta t_{LO}$
$+\Delta t_{LO}$
$+\Delta t_{LO}$
$\tilde{\boldsymbol{u}}_{LO}(t_{end})$
Feature inputs $\eta(\tilde{\boldsymbol{u}}_{LO})$
$f_{ML}(\theta)$
Prediction $\Delta\hat{\boldsymbol{a}}(t)$
ML corrected LO solutions
$\hat{\boldsymbol{u}}_{LO}(x,t) = \sum_{i=0}^{N_{LO}} (a_{i,LO}(t) + \Delta\hat{a}_i(t))\phi_i(x)$

Figure 3.6: Overview of unsteady post-processing correction workflow

advection problem. Based on the performance on the 1D test case, the most promising techniques are applied to the 2D convection of an isentropic vortex, governed by the Euler equations.

3.3.1 Post-Processing Correction with Supervised Learning

In the first unsteady correction approach, the post-processing correction, the low- and high-fidelity trajectories are computed independently, and the correction depicted by black arrows in Figure 3.5 is applied after the simulations have been computed. The process is summarized in Figure 3.6.

The method closely follows the correction approach described for steady simulations in section 3.2.2, with the addition of the temporal dimension. Again, three steps are conducted: dataset generation, ML training, and ML model prediction. During the first phase, the dataset generation, multiple simulations using the low- and high-fidelity discretization are conducted until a specified time t_{end}. This is done across a defined design space. Previously, the design space for steady application was spanned by Mach number M and angle of attack α. For the unsteady application considered, the variables are dependent on the respective test case and are described in detail in chapter 6. It is assumed that $\Delta t_{\mathrm{HO}} < \Delta t_{\mathrm{LO}}$. The time step sizes have to be chosen such that every low-fidelity time step Δt_{LO} is inside the set of high-fidelity time steps. This allows to obtain a truncated high-order solution at every low-order time instance. After the simulations have been conducted, only the high-fidelity solutions $\tilde{\boldsymbol{u}}_{\mathrm{HO}}$ coinciding with the low-order time steps are truncated to the low-fidelity discretization and are used to compute the correction term $\Delta \boldsymbol{a}(t)$ for each element k. Furthermore, the low-fidelity simulations are used to derive the set of input features $\boldsymbol{\eta}(\tilde{\boldsymbol{u}}_{LO})$. As the input features for the unsteady cases vary, they are described in more detail in the respective results sections. There is no need to compute truncation, correction, and features for t_0 if the initial conditions of both low- and high-fidelity simulations are the same. The ML training does not differ significantly as for the one described in section 3.2.2 for steady simulations. The only notable difference is that the number of training instances per simulation is multiplied by the number of conducted low-fidelity time steps. Finally, during the prediction phase, the ML model infers the correction term element-wise at every time step for each simulation previously not present within the design space. The correction is then used to improve the low-fidelity solution in the interval $[t_0 + \Delta t, t_{end}]$.

Diverging trajectories lead to out of distribution data.

The post-processing correction is a convenient and straightforward approach to implement, as the simulations and thus the data collection can be conducted independently of the ML training and the ML correction. However, a drawback of this method is that the corrections may not follow a uniform distribution, as the HO and LO trajectories diverge over time, as exemplarily shown in Figure 3.5. This can create challenges during the ML training, and, more importantly, can pose a major problem to the ML generalization capabilities. For unseen problems during prediction, the distribution of low-fidelity states can differ significantly, especially if the model is employed at time steps not covered during training.

3.3.2 Iterative Correction with Supervised Learning

To mitigate diverging trajectories as encountered in the post-processing correction approach, an alternative method, the iterative correction, is explored. This alternative trajectory is depicted in red in Figure 3.5. The approach aims to achieve a more uniform distribution of correction values over time by applying the difference between the low-fidelity and the truncated high-fidelity solutions after each solver iteration. This allows the next solver step to start from a corrected solution, enabling the low-fidelity simulation to closely follow the truncated one and thereby minimizing the divergence between the different trajectories. The correction is learned using a supervised training method, after trajectories have been unrolled up to a certain time step t_{end}, during which data is collected for the ML training.

Figure 3.7 presents a summary of the workflow. The training phase follows the same procedure as discussed for the post-processing correction and is therefore not repeated. The key differences are found in the data collection and prediction phases. During the dataset generation step, given on the left of Figure 3.7, the feature inputs and correction terms are computed after each low-fidelity iteration for each sample point in the design space. The correction term is immediately used to correct $\tilde{\boldsymbol{u}}_{\mathrm{LO}}$ before the next solver iteration. The corrected solution, which is equal to the truncated solution, is thus used as restart solution for the next solver iteration until the final time step t_{end} is reached.

During the ML model prediction phase, the correction follows again an iterative approach in-between solver iterations for new simulations previously not encountered within the design space. After the first time step, the features are computed and used as input to the trained ML model, which in turn predicts the correction term. The correction is used to alter the low-fidelity solution, finally used as restart to the next solver iteration until t_{end}. For the initial solution, there is again no need to compute a correction.

Autoregression leads to error accumulation.

The expected drawback of the iterative correction, occuring after employing the trained model for longer time periods, is depicted as orange trajectory in Figure 3.8. The predicted correction $\Delta\hat{\boldsymbol{a}}(t)$ does not perfectly match the true correction term $\Delta\boldsymbol{a}(t)$, leaving an error term as given in (3.11). Even if this error term is small, it will accumulate with each time step, potentially leading to non-physical results. This phenomenon, known as data drift, occurs because the statistical nature of the data seen during training differs from that encountered during deployment, resulting in o.o.d. data, as described in section 2.2.3. This is typical for autoregressive models: here, future predictions are dependent on previous predictions. The model is trained in a supervised learning fashion, for which the data has been pre-collected and does not take into account the error term. Since the trained ML model interacts with the solver during the prediction, the correction

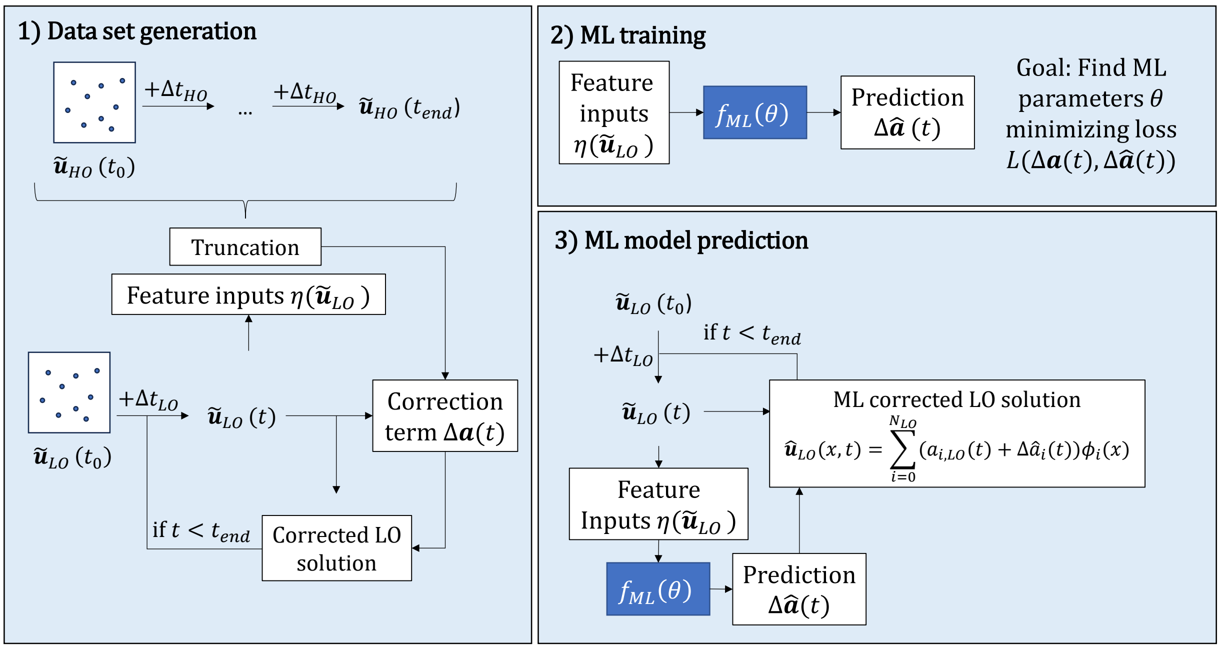

1) Data set generation
$+\Delta t_{HO}$
...
$+\Delta t_{HO}$
$\tilde{\boldsymbol{u}}_{HO}(t_{end})$
$\tilde{\boldsymbol{u}}_{HO}(t_0)$
Truncation
Feature inputs $\eta(\tilde{\boldsymbol{u}}_{LO})$
$+\Delta t_{LO}$
$\tilde{\boldsymbol{u}}_{LO}(t)$
Correction term $\Delta\boldsymbol{a}(t)$
$\tilde{\boldsymbol{u}}_{LO}(t_0)$
if $t < t_{end}$
Corrected LO solution
2) ML training
Feature inputs $\eta(\tilde{\boldsymbol{u}}_{LO})$
$f_{ML}(\theta)$
Prediction $\Delta\hat{\boldsymbol{a}}(t)$
Goal: Find ML parameters θ minimizing loss $L(\Delta\boldsymbol{a}(t), \Delta\hat{\boldsymbol{a}}(t))$
3) ML model prediction
$\tilde{\boldsymbol{u}}_{LO}(t_0)$
if $t < t_{end}$
$+\Delta t_{LO}$
$\tilde{\boldsymbol{u}}_{LO}(t)$
ML corrected LO solution
$\hat{\boldsymbol{u}}_{LO}(x,t) = \sum_{i=0}^{N_{LO}} (a_{i,LO}(t) + \Delta\hat{a}_i(t))\phi_i(x)$
Feature Inputs $\eta(\tilde{\boldsymbol{u}}_{LO})$
$f_{ML}(\theta)$
Prediction $\Delta\hat{\boldsymbol{a}}(t)$

Figure 3.7: Overview of unsteady iterative correction with supervised learning workflow

term influences the next solution, which in turn affects the subsequent prediction. In this work, introducing noise to the model input features is investigated as regularization method, in essence mimicking the error behavior, and has been suggested in previous work for stabilizing time-series predictions [145]. As introducing noise is case and feature specific, the implementation details are discussed in the respective applications in chapter 6.

3.3.3 Iterative Correction with Reinforcement Learning

To account for the actual error of the ML predictions, several studies suggest avoiding the separation of data collection and training, recommending instead to integrate the training process into the CFD solver iterations [7, 41, 162]. This approach enables learning while unrolling the solution trajectories. Many such studies utilize a differentiable solver with adjoints, which allows the feedback from the corrected solution to be incorporated into the minimization of the training loss, leading to more accurate and stable long-term predictions. An alternative approach to a differentiable solver, since many CFD solvers lack adjoint capabilities, is RL. Coupling a CFD solver as an environment which interacts with a RL agent requires no modifications within the solver, while allowing for an online learning approach.

Figure 3.9 summarizes the RL method followed for this work, showing that the data generation step for the low-fidelity simulations is no longer separated from the ML training procedure. Since RL does not require labeled input and output pairs, the high-fidelity data does not necessarily need to comprise the complete truncated high-order data. In theory, anything from full field data to sparse data obtained from measurements can be utilized to formulate the reward function. If expensive high-order simulation data is used, it can be precomputed, as it is not subject to any changes during the training phase. Thus, the truncated higher-

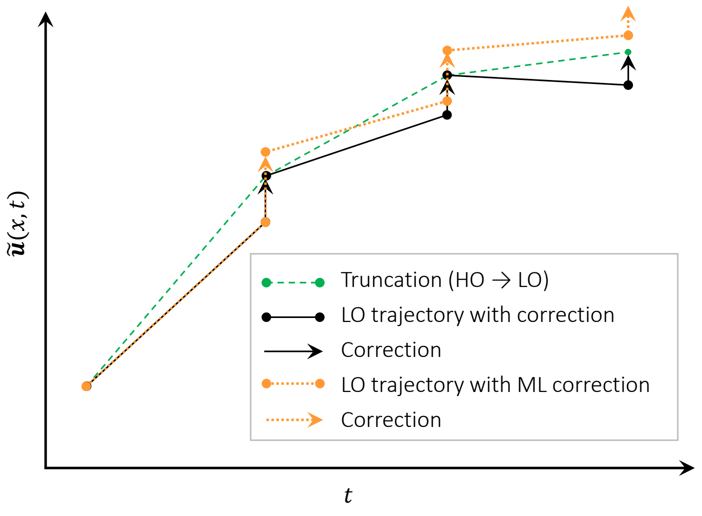

$\tilde{u}(x,t)$
t
Truncation (HO → LO)
LO trajectory with correction
Correction
LO trajectory with ML correction
Correction

Figure 3.8: Data-drift

order solutions at each low-fidelity time step can also be obtained in advance and stored in memory.

Every episode begins by initializing all simulations in the design space at t_0, as depicted on the left of Figure 3.9. As previously mentioned, the specific design spaces vary case by case, and are described in the respective results chapter 6. Initializing multiple simulations at once can be seen as multi-environment RL training, as the agent processes and learns from multiple scenarios at once. Then, the CFD solver iterates over one time step Δt. From the resulting simulations, features are computed, acting as the observation of the environment's state, which are passed to the ML agent. The PPO RL algorithm returns a probability distribution of possible actions based on the received features. This is realized by a defined standard deviation σ, a hyperparameter during training, whereas the actor network predicts the mean μ of the probability distribution. Such an action sampling introduces variance into the training, encouraging exploration in the PPO algorithm. The action for each element k, which is the correction term for the current time t, is returned to the environment, where it is used to correct the previously computed simulation results. This corrected solution serves as restart for the next CFD solver iteration. Furthermore, a reward function is formulated which compares the corrected solution with the ground truth solution, i.e. the truncation at current time step. This procedure is repeated until some stopping criterion is reached, such as the final time t_{end}.

At the end of such a trajectory, tuples of state observations, actions, and rewards have been collected. Note that since the actor predicts a correction for each element k separately, and the observation collection and reward computation oc-

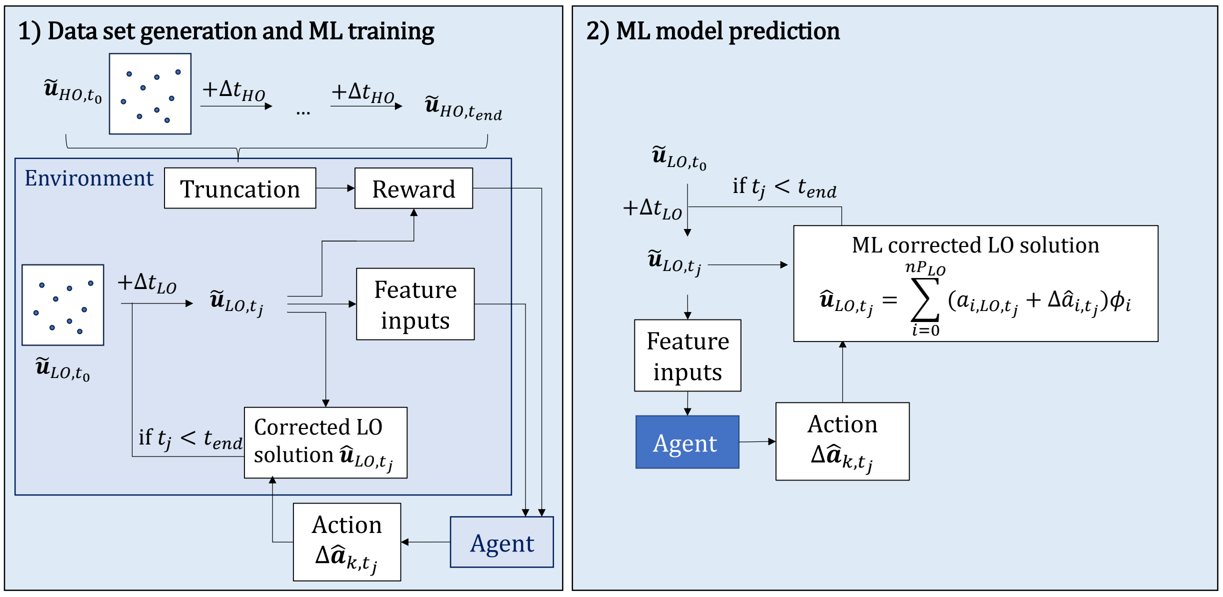

1) Data set generation and ML training
$\tilde{\boldsymbol{u}}_{HO,t_0}$
$+\Delta t_{HO}$
...
$+\Delta t_{HO}$
$\tilde{\boldsymbol{u}}_{HO,t_{end}}$
Environment
Truncation
Reward
$+\Delta t_{LO}$
$\tilde{\boldsymbol{u}}_{LO,t_j}$
Feature inputs
$\tilde{\boldsymbol{u}}_{LO,t_0}$
if $t_j < t_{end}$
Corrected LO solution $\hat{\boldsymbol{u}}_{LO,t_j}$
Action $\Delta\hat{\boldsymbol{a}}_{k,t_j}$
Agent
2) ML model prediction
$\tilde{\boldsymbol{u}}_{LO,t_0}$
if $t_j < t_{end}$
$+\Delta t_{LO}$
$\tilde{\boldsymbol{u}}_{LO,t_j}$
ML corrected LO solution
$\hat{\boldsymbol{u}}_{LO,t_j} = \sum_{i=0}^{nP_{LO}} (a_{i,LO,t_j} + \Delta\hat{a}_{i,t_j})\phi_i$
Feature inputs
Agent
Action $\Delta\hat{\boldsymbol{a}}_{k,t_j}$

Figure 3.9: Overview of unsteady iterative correction with reinforcement learning workflow

cur on an element-by-element basis, this setup is interpretable as a Multi-Agent Reinforcement Learning (MARL) approach. With MARL, each element has its own agent, though in this case they all share the same weights. The collected tuples are used to compute the discounted reward, the resulting advantage function, and both policy and value losses to finally update the parameters of the actor and critic network via backpropagation. Collecting the trajectories and training the agents concludes one episode, and the next begins by re-initializing the training simulations. This cycle of episodes is repeated until a predefined number of episodes is reached or the reward converges.

The inference phase, given on the right of Figure 3.9, follows the same procedure as the iterative correction approach described in the previous section. With PPO, not the whole agent is needed for predictions outside of the training, but only the actor, with which deterministic actions are achieved using the mean of the predicted probability distribution.

Exploration versus exploitation and the challenge of stability, sample efficiency, and reproducibility.

One of the main challenges in employing RL algorithms is finding a balance between exploration and exploitation. For the PPO algorithm, this balance is controlled by the chosen standard deviation and the magnitude of the entropy factor. However, increasing exploitation can lead to instabilities, causing abrupt changes in the agent's behavior. These sudden shifts can result in poor rewards from which the agent cannot recover, leaving it effectively stuck. Sample inefficiency is another major limitation of many RL algorithms, especially for on-policy methods like PPO. Such methods require a large number of episodes to achieve convergence. Finally, reproducibility poses another challenge due to the stochastic nature of the environment, the learning process, or both. Research has shown that minor changes, such as altering the random seed while keeping the model and hyperparameters fixed [73], can introduce sufficient variability to produce significantly different results.

Chapter 4

Application: Correction of Steady Finite Volume Simulations

Performing simulations with the FV method, typically of second order, requires highly resolved computational grids to achieve sufficient accuracy. This increases the number of degrees of freedom as well as the computational cost. Take for example the simulation of a 2D NACA0012 airfoil across different grids, resulting in the lift coefficients and wall clock times reported in Figure 4.1, where the same boundary conditions and hardware has been used for all simulations. The data suggests that neither accuracy nor time scales linearly with the number of degrees of freedom. Increasing the number of elements and with this the number of degrees of freedom increases accuracy, until convergence is reached and a lift coefficient comparable to the reference value is obtained. Obviously, the more accurate the

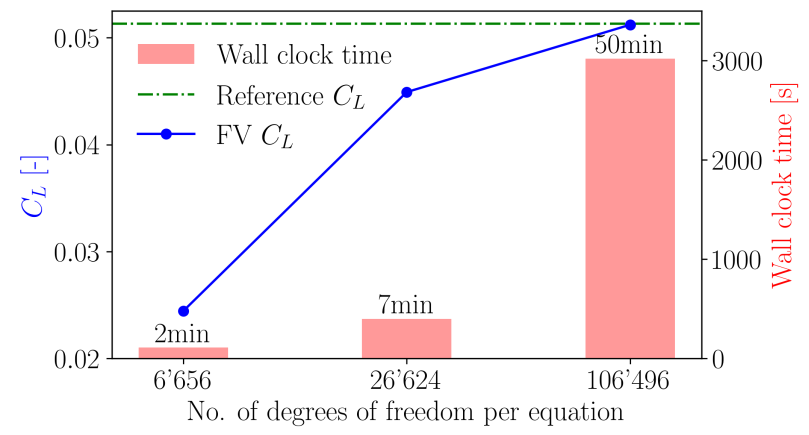


Figure 4.1: Simulation time vs. accuracy for varying grid resolutions

solution is, the more expensive it becomes. Though only shown exemplary on a NACA0012 airfoil, this effect can be observed for many other cases [103, 107, 98].

Classical approaches for achieving fast and accurate simulations focus on reducing the wall clock time required to obtain precise solutions. In this chapter, ML methods are explored that aim for the opposite: enhancing the accuracy of simulations that are already inexpensive, while preserving their low computational cost. For this, it is investigated if ML methods are capable of learning the discretization error of coarse grid solutions, subsequently used as a corrective function. The proposed approach is showcased in this chapter on turbulent flows around the 2D RAE2822 airfoil and the 3D LANN wing. The content of this chapter has previously been published in [85] and in [86]. The method is described in detail in section 3.1.1 and section 3.2.1.

4.1 2D Case - RAE2822 Airfoil

The RAE2822 airfoil geometry is a standard case for numerical investigations [92, 152, 174], since an extensive wind tunnel measurement campaign has been conducted [37]. The RAE2822, shown in Figure 4.2, is a rear-loaded and subcritical airfoil, with a roof-top type pressure distribution at design conditions with $C_L = 0.56$ at $M_{\text{inf}} = 0.66$ and angle of attack $\alpha = 1.06°$ [1]. In this work, the Reynolds number is kept constant at $Re = 5.7 \times 10^6$, similarly to one of the investigated cases of the test campaign. Parametrization of the angle of attack and Mach number within the design space allow for a rich dataset, resulting in continuous flow conditions as well as conditions with discontinuities, specifically shocks.

4.1.1 Data Generation

This section discusses the data generation procedure for the RAE2822 airfoil. This includes the simulation setup and the comparison of coarse and fine grid results, as well as an assessment of using injection as mapping operator $\mathcal{I}_f^c$, as described in section 3.1.1.

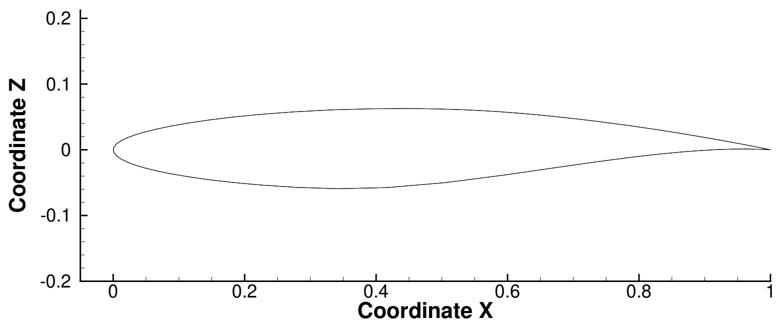


Figure 4.2: RAE2822 airfoil geometry

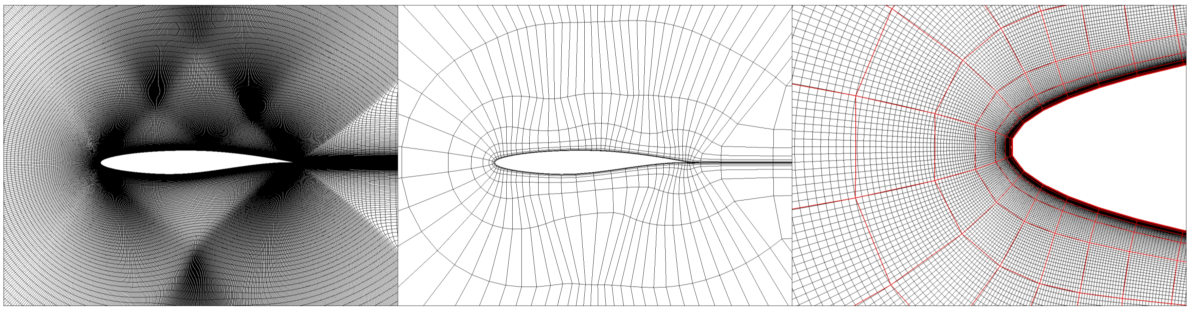

(a) Fine grid (b) Coarse grid (c) Nested grids

Figure 4.3: Nested grids for RAE2822

Simulation Setup

For the simulations, the RANS equations and the negative version of the Spalart-Allmaras turbulence model are employed. The solver employs the FV method with a vertex-centered data structure. Sections of the coarse and fine C-type grids used for this study are shown in Figures 4.3a and 4.3b, respectively. The number of elements and degrees of freedom per equation are reported in table 4.1. A detailed study for the complete grid sequence can be found in [105]. Since the grids are nested and thus the coarse grid nodes are congruent with the respective fine grid nodes, as is shown in Figure 4.3c, injection can be used as the mapping operator $\mathcal{I}_f^c$. With injection, for each coarse grid node the value of the corresponding fine grid node is selected.

Steady-state simulations are performed between Mach number 0.52 and 0.82 and angles of attack between 0.0° and 9.0°. In total, 60 simulations are conducted for the training and validation set, as given in Figure 4.4. Half of the samples are located in the regime in which the lift coefficient behaves linearly, and the other half in the area with high angles of attack, where the lift coefficient is expected to behave non-linearly. The separation of these two areas is indicated in Figure 4.4 by the dashed line. This ensures that a higher sample density is achieved in the latter, which is assumed to be more difficult to learn due to the presence of discontinuities. The simulation points are generated with a design of experiment method, specifically a Latin Hypercube and a Halton sampling. The different sampling strategies have been chosen to achieve higher variance within the dataset. For the test set, additional 17 and 12 simulations are conducted at

Table 4.1: Number of elements, degrees of freedom per equation, and average wall clock time for FV study, RAE2822

	Elements	**Degrees of freedom**	**Time**
Coarse	1'280	1'368	1 min 30 s
Fine	327'680	329'088	7 h

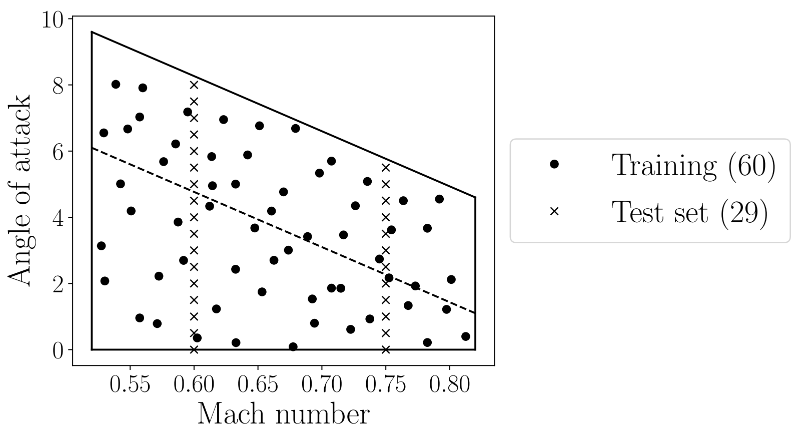


Figure 4.4: Sampling points for training and test sets, where the dashed line divides the different sampling areas, RAE2822

Mach numbers 0.6 and 0.75, respectively. To retrieve the complete data base, for each sample coarse and fine grid simulations are conducted. For all coarse and fine grid computations, a reduction of 12 orders of magnitude of the density residual is obtained. The fine grid simulations are computed on a cluster, using one node with eight processes, while the coarse grid equivalents are realizable on a desktop computer. The respective average wall clock time is reported in table 4.1. These times indicate that using only the coarse grid with subsequent correction would greatly lower the required computational time, since the coarse grid simulations converge rapidly compared to the fine grid equivalents needing several hours. Nevertheless, this also indicates the disadvantage of purely data-driven methods: obtaining extensive datasets for the training of ML models is an expensive task.

Simulation Results

Before the training of any data-driven model, a comparison between coarse and fine grid simulation is provided in Figure 4.5, for a sample point within the design space, at Mach number $M = 0.75$ and angle of attack $\alpha = 5.5°$. The plot shows the pressure coefficient C_P for both, coarse and fine grid simulation, and additionally the C_P distribution from the fine grid solution after being mapped to the coarse grid discretization using injection. The coarse grid solution does not match the fine grid solution, especially in the vicinity of the shock location. Mapping the fine grid solution to the coarse grid leads to a well preserved C_P distribution. Only at the shock location, interpolating the values between the limited number of surface elements is not sufficient to match the fine grid pressure coefficient. Nevertheless, the shock is better approximated than with the coarse grid solution. Thus, it can be concluded that the number of degrees of freedom on the coarse grid is sufficient to capture this value when using the injection operator. In contrast, conducting a simulation on this low-fidelity discretization is not sufficiently accurate.

This is also visible when comparing the coarse and fine grid results with mea-

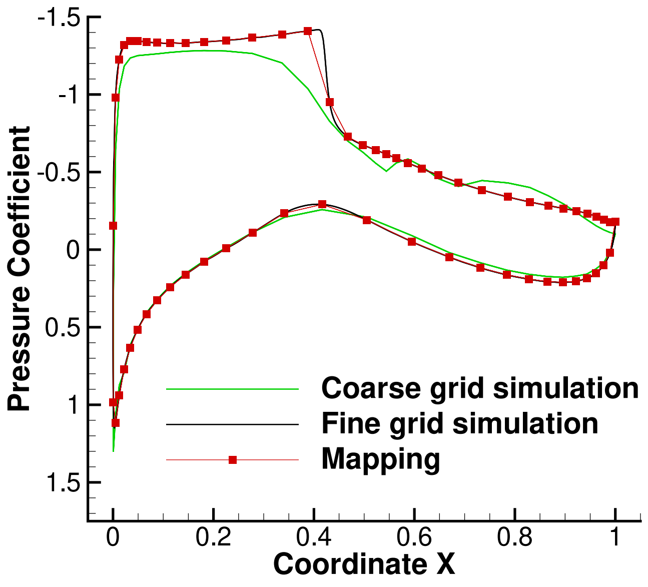


Figure 4.5: Pressure coefficient C_P for coarse, fine, and mapped solution, at $M = 0.75$ and $\alpha = 5.5°$, RAE2822

Table 4.2: Lift coefficient C_L for coarse and fine grid simulation, reference values and test campaign, at $M = 0.676$ and angle of attack $\alpha = 1.93°$, RAE2822

	C_L	Lift count difference
Coarse	0.451	115
Fine	0.557	9
Ref. 1 [105]	0.569	3
Ref. 2 [105]	0.561	5
Test campaign [1]	0.566	-

surement and reference data. For this, simulations are conducted at $M = 0.676$ and $\alpha = 1.93°$, for which the lift coefficient C_L from a test campaign [1] and reference simulation data from literature [105] is available. Table 4.2 depicts the lift coefficient of these sources as well as the lift coefficient derived from coarse and fine grid simulation. Additionally, the table reports the number of lift counts between the simulation and the test campaign value, for which one lift count corresponds to 0.001. The table highlights the inaccuracy of the coarse grid solution, as the lift count is significantly higher than for the fine grid solution and the reference values. The fine grid solution displays a relatively low deviation to the references and the test campaign, such that the simulations obtained with this grid and the corresponding settings are deemed accurate enough for the following study.

4.1.2 Machine Learning Correction

In this section, the ML model training is presented. Furthermore, the ML correction results for the test sets are shown, while comparing the performance of all three ML models.

Training Setup

All three ML models introduced in 2.2.2 are used to learn the correction term for the RAE2822 case. These include a RF, NN, and GNN model. With 60 simulations in the training and validation set and 1'368 degrees of freedom per simulation, this amounts to 82'080 training instances. As inputs, given in (3.15), the models receive a local cell Reynolds number, as given in (3.14), and the first and second derivative of the variables of interest $\tilde{\boldsymbol{u}}$.

The trained models return one correction term for each of the variables of interest to be corrected. These are $\tilde{\boldsymbol{u}} = [\rho, \boldsymbol{U}, P]^T$, including density ρ, velocity vector $\boldsymbol{U} = [v_x, v_z]^T$, and pressure P. Note that the turbulent variable $\tilde{\nu}$ is not corrected, as this term is not necessary to derive variables of practical interest, such as for example the lift coefficient C_L. Nevertheless, adding features which consider the turbulent nature of the flow increases the accuracy. This is done through the local Reynolds number, entailing the laminar and turbulent viscosity. It was found that adding certain global values as features, such as the angle of attack and Mach number, had no significant influence on the validation metrics. Other grid element related values, such as the skewness or cell volume, had also no relevant effect as features. Only the cell Reynolds number, which includes the wall distance, leads to improved validation metrics. This results in a total of 25 input features and 4 outputs per vertex. Standardization to zero mean and unit variance is applied to each in- and output variable.

A four-fold cross validation is performed to train the models. During each fold, the models are trained on 45 simulations while being validated on the remaining 15. Additionally, Bayesian hyperparameter optimization with TPE is used to explore different model parameters. The remaining 39 simulations at Mach numbers $M = 0.6$ and $M = 0.75$ are then tested only on the best performing models resulting from this cross validation. The final hyperparameters found during this procedure are reported in tables 4.3, 4.4, and 4.5. For both the NN and GNN, The batch size has shown no significant influence in the model performance, while mainly affecting training speed. The learning rate decays exponentially up to a fifth of the initial value after 1'000 epochs and a stopping criterion based on the validation data is employed for both network based models. The Adam optimizer leads to improved performance compared to Stochastic Gradient Descent. Similarly, the hyperbolic tangent as non-linear function outperforms the rectified linear unit function.

Table 4.3: RF hyperparameters, RAE2822

Hyperparameter	**Value**
No. of decision trees	236
Features per split	12

Table 4.4: NN hyperparameters, RAE2822

Hyperparameter	**Value**
No. of layers	7
No. of neurons per layer	317
Initial learning rate	3.24×10^{-4}

Table 4.5: GNN hyperparameters, RAE2822

Hyperparameter	**Value**
No. of layers	9
Feature dimension per layer	285
Initial learning rate	1.62×10^{-3}

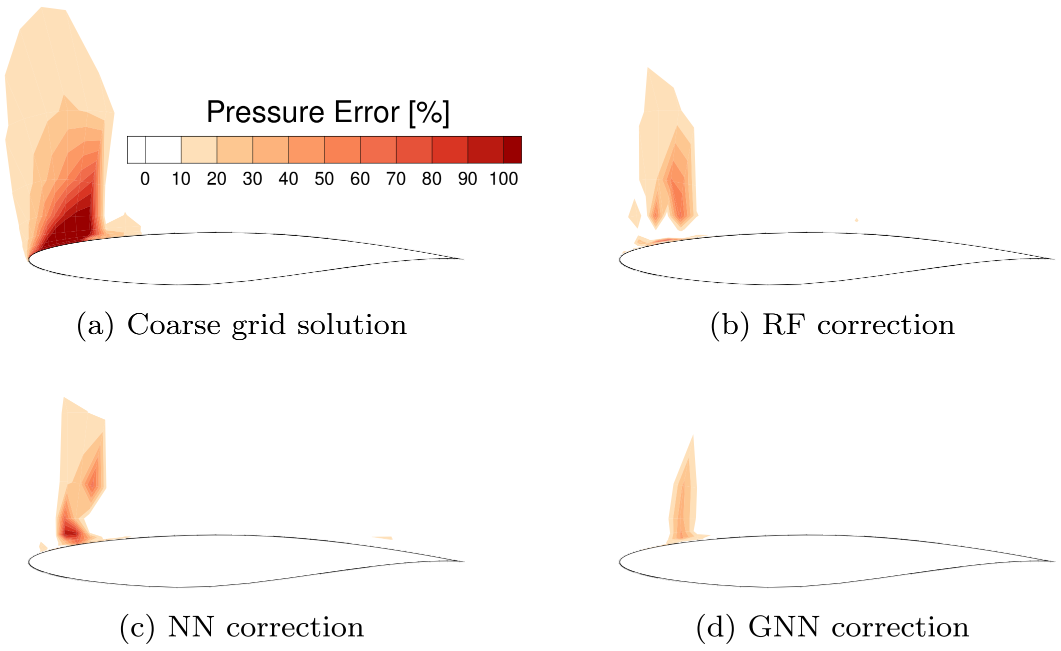


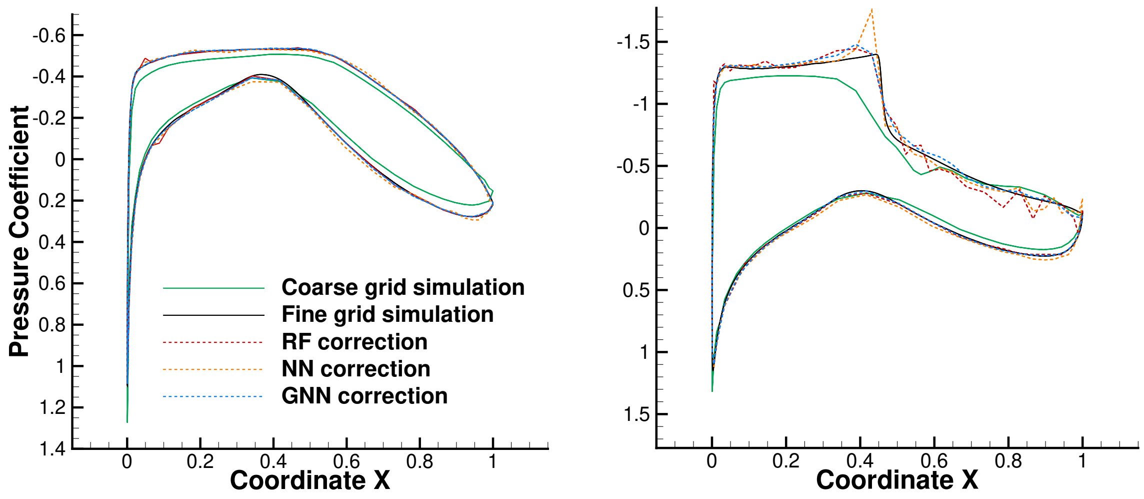
Figure 4.6: Pressure error in percentage compared to high-fidelity solution projected to coarse grid at $M = 0.6$ and $\alpha = 7.5^\circ$, RAE2822

All described ML model trainings are performed either on CPU for the RF, or on GPU for the deep learning models. The GPU model is an NVIDIA Quadro P2200 with 5 GB memory. The wall clock times are around 20 minutes for the RF training, 1.7 hours for the NN training, and 2.2 hours for the GNN training. The deployment of the models during the prediction phase, e.g. retrieving the corrections, takes only seconds.

Correction Results

In this section, the test set at Mach numbers $M = 0.6$ and $M = 0.75$ are presented. Since the correction of the flow field is the main focus of the proposed method, firstly a qualitative comparison of the pressure error is given in Figure 4.6 for Mach number $M = 0.6$ at angle of attack $\alpha = 7.5°$. Figure 4.6a illustrates the error between the coarse grid simulation and the high-fidelity solution after injection. It is clearly visible that the error is mainly present in front of the shock location. Figures 4.6b, 4.6c, and 4.6d show the reduced error after the correction of the three ML models. All three models are able to reduce the error between the leading edge and the shock location, thus predicting the shock where the coarse grid simulation fails to do so. All models succeed in reducing the error, with the GNN outperforming the RF and NN. For the RF model the maximum error is found at 85.6 % within the boundary layer, for the NN it is found outside the boundary layer but with an amount of 103.1 %. The GNN model decreases the error to a maximum of 48.1 % and the resulting pressure field shows the most accurate pressure distribution. This might be explained by the GNN taking into account neighboring information of each corrected vertex value, leading to a regularized and smooth correction, whereas the other models exhibit noisy behavior as they take into account local vertex information only. Similar correction trends are observed for the other variables and flow conditions.

In the following, various surface and integral coefficients are presented and compared to quantitatively assess the corrections. It is emphasized that all coefficients are computed from either the high- or low-fidelity simulation, or from ML corrected field solutions. In Figure 4.7, the pressure coefficient C_P along the airfoil surface is shown for the coarse and fine grid simulation, as well as the ML corrections for Mach number $M = 0.6$ at angle of attack $\alpha = 0.5°$ and for Mach



(a) $M = 0.6$ and $\alpha = 0.5°$ (b) $M = 0.75$ and $\alpha = 5.0°$

Figure 4.7: Pressure coefficient C_P for two test set samples, RAE2822

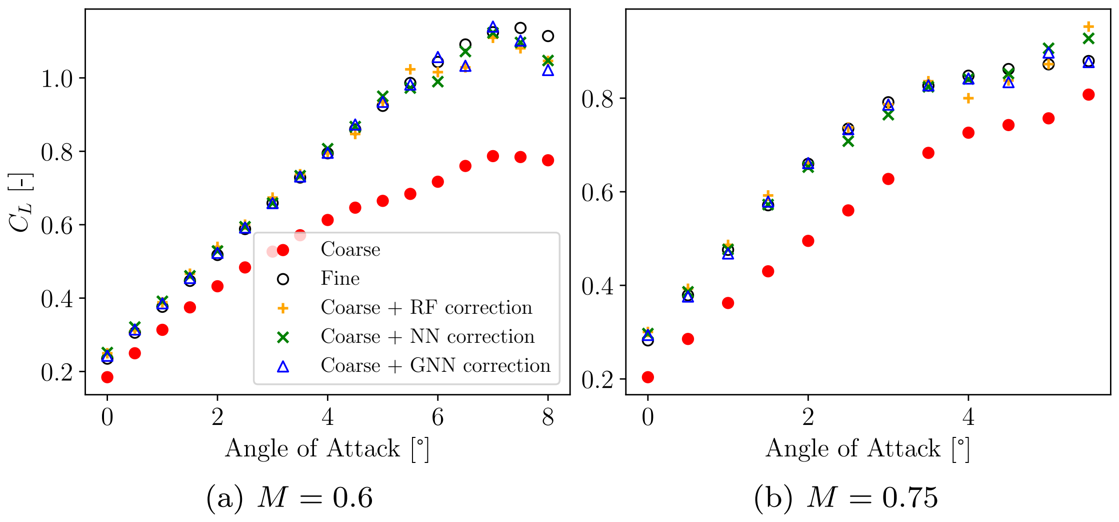


Figure 4.8: Lift coefficient C_L for test set samples, RAE2822

number $M = 0.75$ and angle of attack $\alpha = 5.0°$. At low angle of attack, all models perform well. Only the RF shows some outliers on both upper and lower surface around $x = 0.1$. For the example at higher Mach number and higher angle of attack, the GNN correction approximates the distribution well, while both the RF and the NN corrections show outliers or oscillating behavior, especially on the suction side.

Figure 4.8 presents the resulting lift coefficients C_L across all angles of attack for both test sets. All models improve the accuracy significantly and approach the lift coefficient of the fine grid simulation, especially for lower angles of attack. At higher angles of attack, a reduction of the prediction accuracy occurs. This difficulty can be attributed to the discontinuities encountered here, since shocks are present at these conditions.

A final comparison between the models is done by computing varying metrics, defined in 2.2.4, to assess quantitatively the accuracy of the predicted discretization error. These metrics include the coefficient of determination R^2 (2.41), the MAE (2.40), and the MSE (2.39). Figures 4.9 and 4.10 depict these scores averaged over all simulations for each test set. The metrics are computed between the scaled true correction term $\Delta\tilde{u}$ and the scaled predicted correction term $\Delta\hat{u}$ for each variable, i.e density ρ, velocities v_x and v_z, and pressure P. The figures indicate that the GNN is the best performing model across all metrics and variables for both test sets. The model reaches greater R^2 scores, while showing the lowest values for MAE and MSE. Considering the test set at lower Mach number $M = 0.6$ in Figure 4.9, the NN and the GNN show a consistent performance with an R^2 value between 0.9 and 0.97 across all corrections, whereas the RF shows a drop of performance for pressure and density. For the results for the test set at Mach number $M = 0.75$, given in Figure 4.10, all models show a performance degradation for the correction of the pressure and density field for the R^2 values. Again, the RF model scores lowest for R^2 and highest for MAE and MSE values.

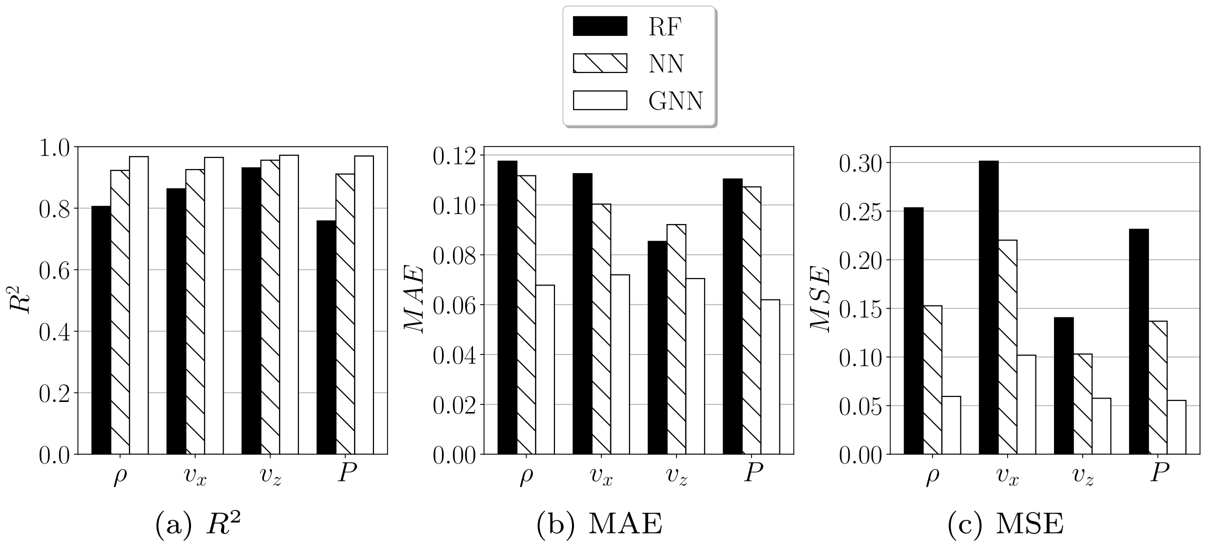


Figure 4.9: Test set metrics for $M = 0.6$, RAE2822

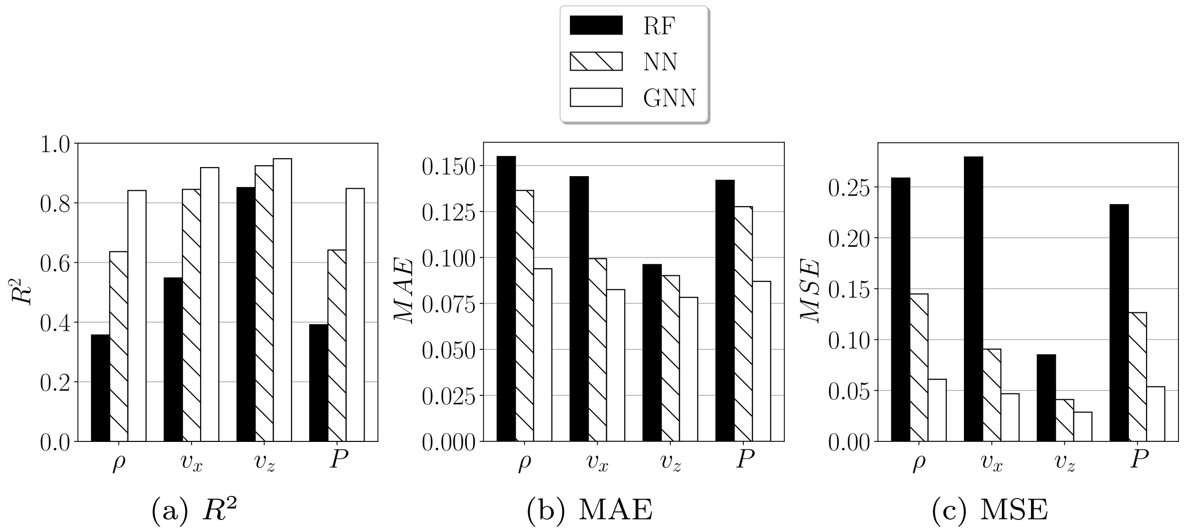


Figure 4.10: Test set metrics for $M = 0.75$, RAE2822

The significant decrease of the coefficient of determination for the RF model for this test set indicates that it is not as capable of predicting accurate corrections for flows with discontinuities as the NN or GNN. The GNN shows for all variables a relatively high R^2 score above 0.84. The test set metrics of the NN and GNN are observed to be in similar ranges as the ones obtained during cross validation for the validation data. Only the RF shows a significant loss of corrective capability for the test set at high Mach number, indicating an overfit to the training and validation sets.

4.2 3D Case - LANN Wing

Extending and testing methods on 3D cases is of importance to showcase scalability to industrial relevant problems. Therefore, turbulent flow around the LANN wing geometry is considered. The geometry is a supercritical wing, and although relatively simple, it has proven to be a valuable test case for many CFD solvers, since an extensive wind tunnel test campaign has been conducted [54]. The LANN wing is designed as a moderate-aspect-ratio transport wing configuration, at design conditions $M_{\text{inf}} = 0.82$ and $C_L = 0.53$. Similarly to the previously considered 2D test case, steady simulations are considered at a constant Reynolds number of 7.3×10^6, while Mach numbers and angles of attack vary.

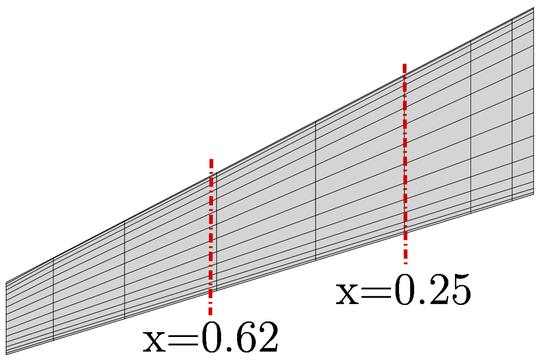
4.2.1 Data Generation

In the following, the simulation setup to generate data for the LANN wing is presented. This also includes a comparison between coarse and fine grid simulation results, as well as the injected high-fidelity solution. For this, a sample point is chosen for which experimental data is available. This allows to validate the simulation settings and confirm that the fine grid simulation is sufficiently accurate.

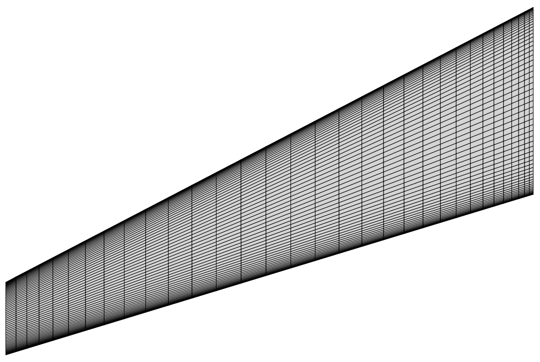
Simulation Setup

The coarse and fine grid for the LANN geometry are obtained from a nested grid series [31], allowing to use injection as mapping operator $\mathcal{I}_f^c$, as described in section 3.1.1. The coarse and fine grid of the upper surface area are shown in Figures 4.11a and 4.11b, respectively, and the resulting number of elements and degrees of freedom per equation are reported in table 4.6.

As for the 2D case, the RANS equations with the negative Spalart-Allmaras turbulence model are used. In total, 40 simulations are computed between Mach



(a) Coarse surface grid



(b) Fine surface grid

Figure 4.11: Nested grids for LANN wing

Table 4.6: Number of elements, degrees of freedom per equation, and average wall clock time, LANN

	Elements	**Degrees of freedom**	**Time**
Coarse	7’040	8’236	7 min 40 s
Fine	450’560	469’213	5 h 10 min

number 0.52 and 0.82 and angle of attack 0.0° and 7.5°, as presented in Figure 4.12. Two design of experiments are carried out and combined to obtain a denser distribution within the flow regime along high angles of attack. A Latin Hypercube sampling is conducted to generate 20 data points within the bottom area and a Halton sampling to generate 20 data points within top area. This results in 20 less simulations for the training set compared to the 2D test case. Nevertheless, each 3D sample point consists of more degrees of freedom and thus more training instances. For testing, 14 points at Mach number $M = 0.6$ and 11 points at Mach number $M = 0.75$ are generated. For each sample point a coarse and fine grid simulation is computed with convergence criterion of 12 orders of magnitude for the density residual. The high-fidelity simulations are conducted on a cluster, using one node with eight processes. The coarse grid equivalents are decomposed into four domains and computed on a desktop computer. The respective averaged wall clock time is reported in table 4.6. This emphasizes that at the cost of decreased accuracy, potentially cheaper and more accessible hardware can be employed with a significant decrease in time when relying on low-fidelity simulations.

Simulation Results

Data from experiments are available at Mach number 0.82 and angle of attack 0.6° [183]. Thus, both low- and high-fidelity simulations are computed at this point to compare their accuracy with respect to the measurement data. This is done in Figure 4.13, by plotting the pressure coefficient along the wing surface at

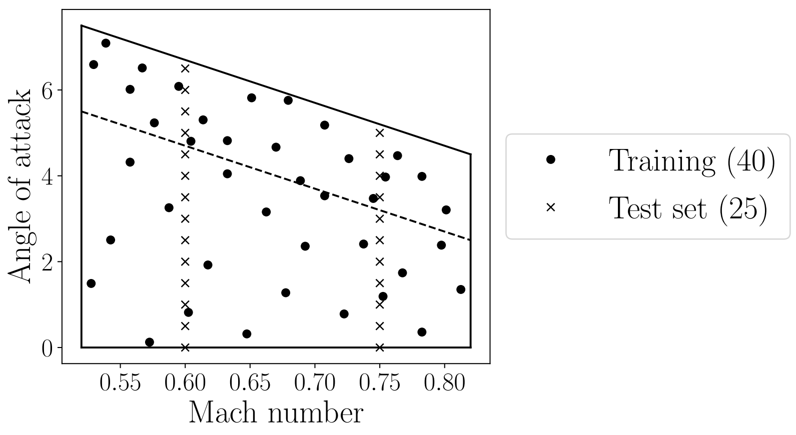


Figure 4.12: Sampling points for training and test sets, LANN

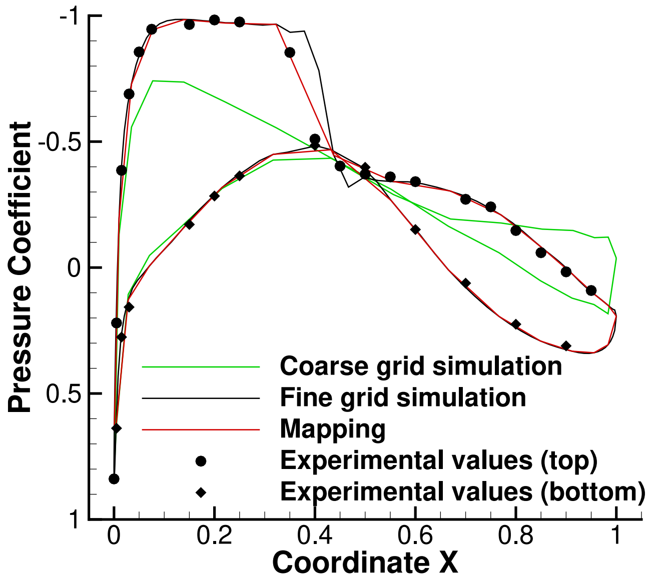


Figure 4.13: Pressure coefficient C_P for coarse, fine, and mapped solution, at $M = 0.82$ and $\alpha = 0.6°$, slice at $x = 0.62$, LANN

the sectional cut at $x = 0.62$. The location of the cross section is indicated in Figure 4.11. The fine grid simulation very well matches the experimental data, with slight deviations around the shock location. Nevertheless, it is by far more accurate than the solution obtained on the low-fidelity discretization: the coarse grid simulation is not capable of capturing the shock. Additional to the experimental value and simulation data, Figure 4.13 also plots the fine grid simulation injected to the coarse grid. The preserved pressure coefficient on the coarse grid discretization shows to be sufficiently accurate, although it slightly deviates from the fine grid solution at the shock location. Thus, it is a reasonable goal to predict corrections for the coarse grid simulation to approximate its respective fine grid counterpart.

4.2.2 Machine Learning Correction

This section describes the conducted ML model training, mainly based on the previously obtained results for the 2D airfoil. Subsequently, the ML corrected coarse grid solutions for the test set are presented, first qualitatively by depicting the percentage error of the pressure field and the pressure coefficient on the wing surface, and then quantitatively, by evaluating the lift coefficient and different metrics across the test samples.

Machine Learning Setup

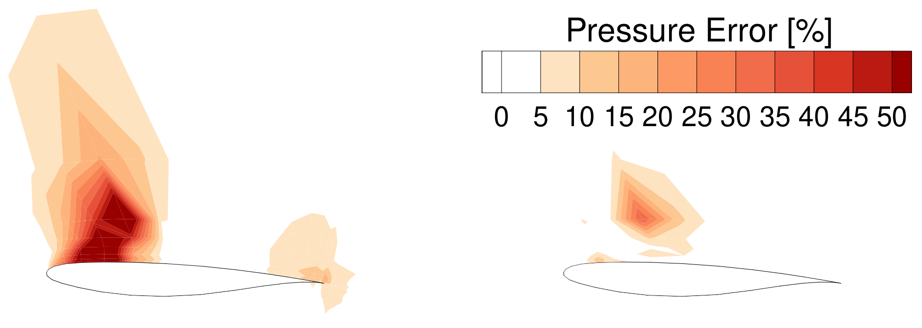
Since cross validation and Bayesian optimization involve rigorous training of several models to find optimal hyperparameters, this is avoided for the 3D case.

Instead, the best performing model from the 2D case is adopted, namely a GNN model, and the same hyperparameters are used to train it on the LANN wing training data. Only the number of epochs deviate from the previous case, since a stopping criterion based on the convergence of the training loss is employed. The training on the 3D case requires more epochs until convergence. The same features are used as inputs, including the derivatives in the third dimension. With the local cell Reynolds number and all three velocity components, this results in a total of 61 input features. As for the outputs, there are a total of five corrections to be predicted. Again, standardization to zero mean and unit variance is applied to all features and outputs.

Compared to the 2.2 hours training time on the two dimensional RAE2822, the GNN training for the LANN wing dataset took about 9.6 hours on the same hardware. This can be put into relation with the additional number of feature inputs and outputs for a three dimensional case, the increased amount of epochs needed, and the increased amount of training instances: on the one hand, the RAE2822 training is conducted on 60 simulations with 1'368 degrees of freedom each, resulting in 82'080 training instances. On the other hand, the LANN wing training data consists of 40 simulations with 8'236 degrees of freedom, resulting in 329'440 training instances. Even if this number increases, an advantage of the correction of low-fidelity simulation is that the number of training instances is still fairly low compared to other deep learning applications, as the training is performed on the coarse grid data. Thus, no additional arrangements have to be taken into account to fit the three dimensional data into memory, as the GPU with 5 GB memory combined with a small batch size is sufficient for this 3D case.

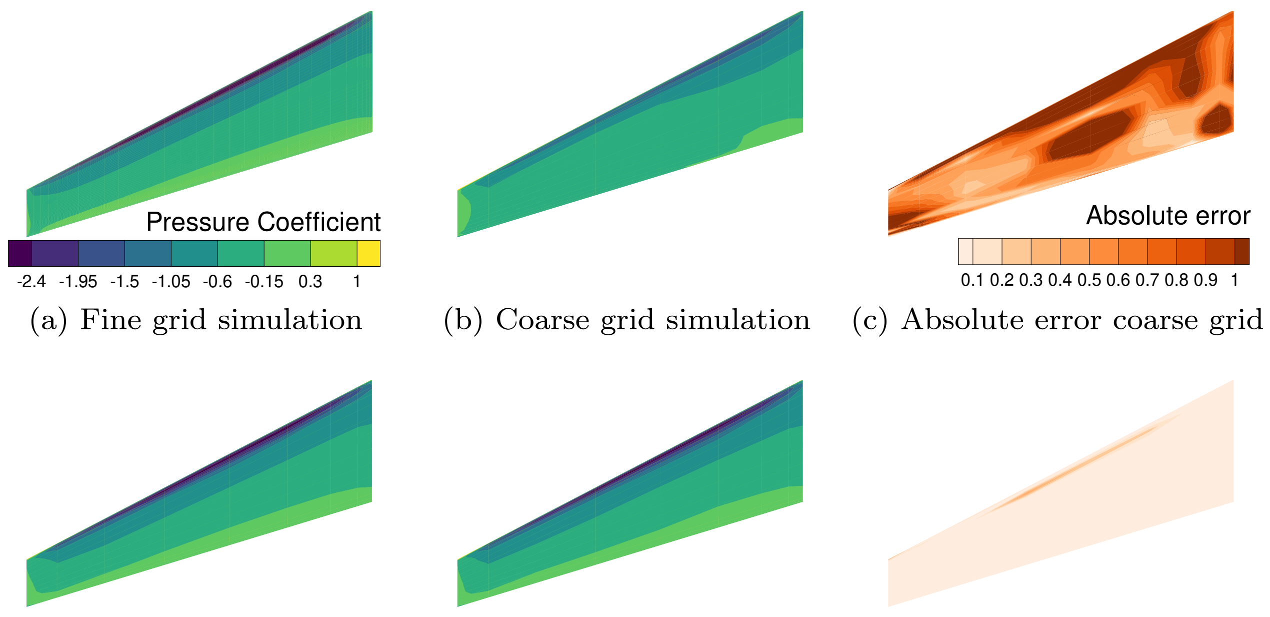
Machine Learning Correction

ML correction results are presented in Figure 4.14 for the LANN wing test sample at Mach number $M = 0.75$ and angle of attack $\alpha = 4.5°$. The figure shows the percentage error of the pressure field for a cross section at $x = 0.25$. The location



(a) Coarse grid solution (b) GNN Correction

Figure 4.14: Pressure error in percentage compared to high-fidelity solution projected to coarse grid at $M = 0.75$ and $\alpha = 4.5°$, LANN



(a) Fine grid simulation (b) Coarse grid simulation (c) Absolute error coarse grid

(d) Projection (e) GNN correction (f) Absolute error GNN

Figure 4.15: Pressure coefficient C_P and absolute error compared to high-fidelity solution projected to coarse grid along upper wing surface at $M = 0.6$ and $\alpha = 5.0°$, LANN

of this cross section is indicated in Figure 4.11. In Figure 4.14a, the error of the coarse grid simulation with respect to the injected high-fidelity solution is depicted. Figure 4.14b presents the corresponding error for the GNN corrected simulation. For the low-fidelity simulation, the error is mainly located in front of the shock area and additionally around the trailing edge. The GNN correction improves the solution, with no error being visible around the trailing edge and a significant error decrease in front of and at the shock location compared to the coarse grid simulation result.

The performance of the GNN is also reflected in the accuracy of the corrected pressure coefficient C_P. This value is presented in Figure 4.15 for the upper wing surface for the test sample at Mach $M = 0.6$ and angle of attack $\alpha = 5.0°$. Figures 4.15a and 4.15b show the pressure coefficient for the fine and coarse grid simulation, respectively. Figure 4.15d presents the pressure coefficient C_P derived from the injected high-fidelity simulation, while Figure 4.15e presents the GNN corrected solution. It is first of all visible that the injection captures well the pressure distribution obtained from the fine grid simulation, although slight deviations can be seen for example at the wing tip. Looking at the coarse grid solution, it is clear that the low-fidelity simulation does not capture an accurate pressure coefficient distribution, especially along the leading edge. The ML correction given in Figure 4.15e notably increases the accuracy, leading to a pressure coefficient distribution which visually well approximates the corresponding distribution from

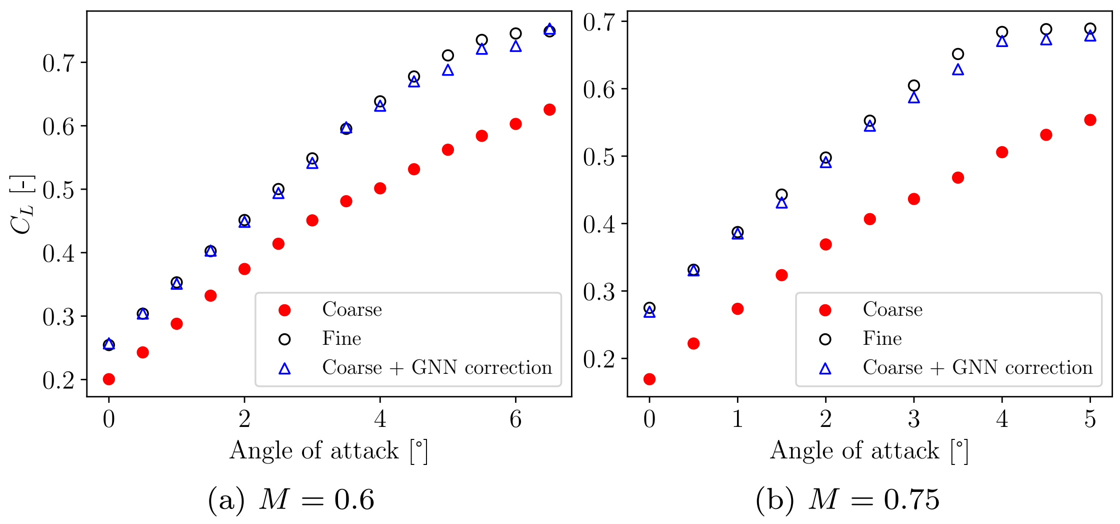


Figure 4.16: Lift coefficient C_L for test set samples, LANN

the injected solution. This is indeed well presented in Figures 4.15c and 4.15f, showing that the absolute error of the C_P distribution is significantly decreased after the GNN correction.

For a quantitative assessment, the lift coefficient is depicted for all test samples in Figure 4.16. Figure 4.16a and Figure 4.16b show the lift coefficients derived from the coarse grid simulation, the fine grid simulation, and the GNN corrected solution across all angles of attack for Mach $M = 0.6$ and at Mach $M = 0.75$, respectively. It is visible that the low-fidelity discretization significantly deviates from the fine grid simulation. At low angles of attack, there is a constant offset between the lift coefficients, while at higher angles of attack, where discontinuities occur, this deviation increases. Similarly to the 2D test case, the GNN correction significantly improves the accuracy of the lift coefficient, and even exhibits less oscillatory behavior at high angles of attack compared to the RAE2822. Nevertheless, the trend of having better improvements at low angles of attack than at higher ones is still visible.

Finally, Figures 4.17 and 4.18 present the evaluation of the coefficient of determination R^2, the mean absolute error MAE, and the mean squared error MSE, averaged over all angles of attack for the predicted corrections, for the test set samples at Mach $M = 0.6$ and at Mach $M = 0.75$, respectively. The coefficient is computed between the scaled true correction term $\Delta\tilde{u}$ and the scaled predicted correction term $\Delta\hat{u}$ for each variable, that is density ρ, all three components of velocity v_x, v_y, and v_z, as well as pressure P. The GNN scores a value of $R^2 = 0.89$ or higher for both test sets, showing the lowest score for the correction for the density variable. Looking at the MAE and MSE, it is visible that larger errors are found for the velocity v_z, and that the MSE increases for the density predictions for the high Mach number samples. Nevertheless, the errors are in general low and their range is comparable to the achieved 2D RAE2822 metrics. While these scores indicate good predictive capabilities, increased performance may be achiev-

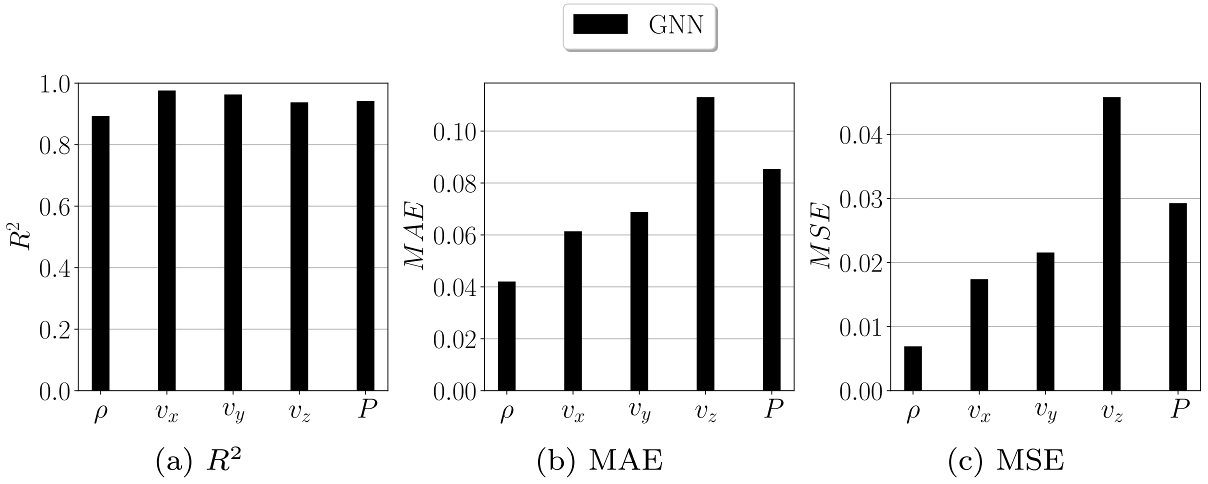


Figure 4.17: Test set metrics for $M = 0.6$, LANN

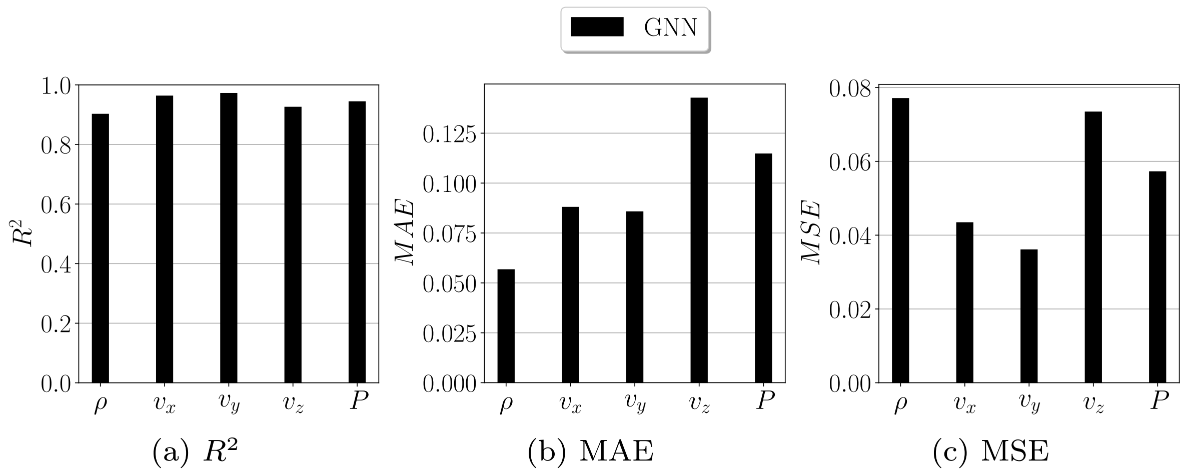


Figure 4.18: Test set metrics for $M = 0.75$, LANN

able by repeating the hyperparameter tuning process. Nonetheless, it has to be highlighted that the ability to apply the same GNN architecture as used in the 2D case suggests that prior experience in setting up ML models and hyperparameters can be effectively transferred to new cases.

4.3 Limitations

The previously presented results highlight the correction capabilities of the trained models, especially the GNN. Nevertheless, the proposed correction approach exhibits several limitations. First, limitations posed by the coarse grid discretization itself are discussed in the following. Secondly, restrictions given by the ML model are examined, focusing on its generalization capacity, the cost of collecting training data, and the computational expenses associated with training the model.

4.3.1 Low-Fidelity Discretization: Coarse Boundary Layer

As previously discussed in section 3.2.1, the ML corrected coarse grid solution approximates in the best case the fine grid solution projected onto the coarse grid discretization, not the fine grid solution itself. Thus, the choice of the coarse grid resolution plays a major role not only in reducing computational cost but also in the maximum achievable accuracy. The previously presented results showed that values such as pressure and lift coefficient are well preserved on the coarse grid. For these values, the ML corrected solutions indeed approximate the high-fidelity solution. Nevertheless, certain values of interest, being accurate on the fine grid, are naturally degraded on the coarse grid discretization. For external aerodynamics, this includes values depending on velocity gradients computed at the boundary. This holds true for both 2D and 3D test case, but it is exemplarily presented here only for the RAE2822 case.

Figure 4.19 depicts the friction coefficient C_f along the airfoil surface for Mach $M = 0.75$ and angle of attack $\alpha = 5.0°$. It shows that the coarse grid simulation is again far from the corresponding fine grid simulation. The coefficient derived from the GNN corrected fields improves this, but it is not capable of achieving the same accuracy as the high-fidelity simulation, leaving a significant gap. Instead, the correction approximates only the coefficient of the fine grid solution injected onto the coarse grid given in red.

While decreasing the number of degrees of freedom by coarsening the grid allows for fast evaluations, it obviously degenerates the accuracy. For the investigated cases, this effect is observable for friction related values. The source of this error becomes obvious when looking at the definition of the wall shear stress τ_w, given in (2.46). The wall shear stress is defined by the velocity gradient perpendicular to the wall. Employing fine grids, the elements along the wall boundary

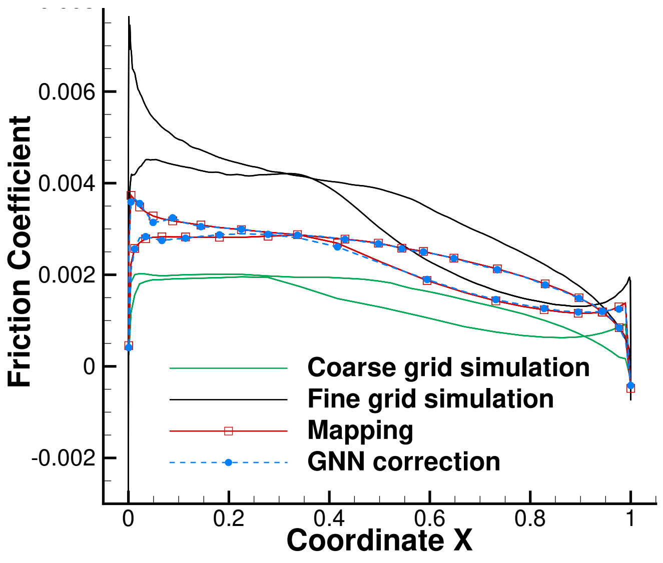


Figure 4.19: Friction coefficient C_f for test sample at $M = 0.75$ and $\alpha = 5.0°$, RAE2822

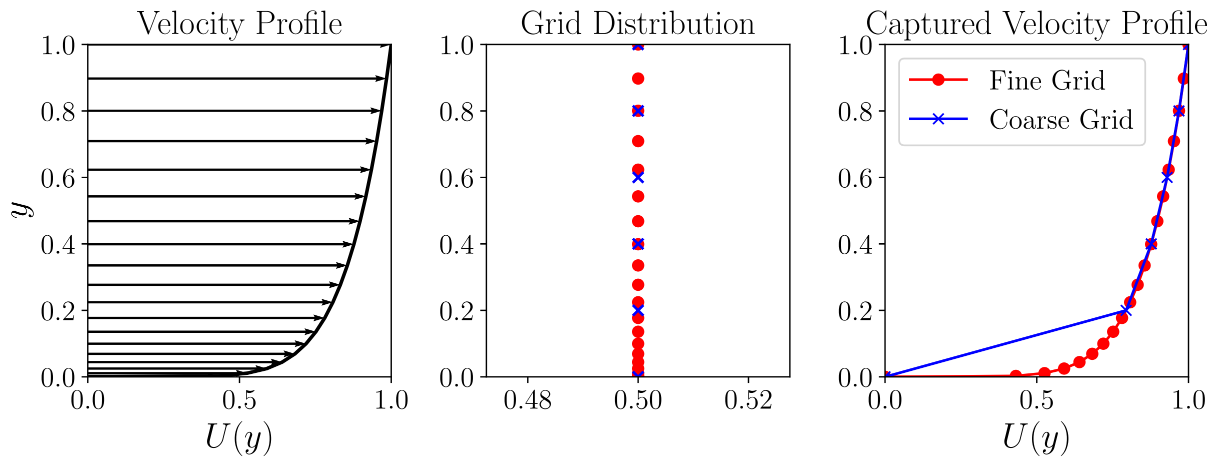


Figure 4.20: Velocity profile $U(y)$ and gradient approximation with coarse and fine grid

are sufficiently thin, and thus capable of accurately capturing velocity gradients. This accuracy is degraded on the coarse grid, where elements along the airfoil wall increase in thickness, on which gradients are crudely approximated. This problem is visually explained in Figure 4.20, presenting a velocity profile $U(y)$ along a wall on the left. It shows how an equidistant coarse grid is not capable of capturing the gradient, while a grid refined towards the boundary does. To alleviate this issue, one could tailor the coarse grid to be finer in the vicinity of walls, while still employing relatively coarse elements elsewhere. Additionally, wall functions could be applied to compute more accurate gradients.

4.3.2 Data-Driven Modeling Restrictions

Additional limitations of the proposed method arise due to the data-driven nature of ML models. Firstly, restrictive generalization capacities are showcased, determining the model's performance on out of distribution data. Secondly, the process of collecting training data is often resource intensive, as acquiring high-fidelity simulations is a time consuming task. The computational expenses associated with training the model, including the need for powerful hardware and extended training times, further add to these limitations. Together, these factors highlight the balance between achieving accuracy and managing practical constraints in the development of data-driven solutions.

Generalization Capabilities

The test samples for the RAE2822 airfoil and the LANN wing, given in sections 4.1.1 and 4.2.1, respectively, have been chosen such that they are located inside the borders of the design space, allowing the trained ML model to interpolate within this space. As described in section 2.2.3, data-driven models cannot be expected to be applicable to new samples out of the distribution of the training

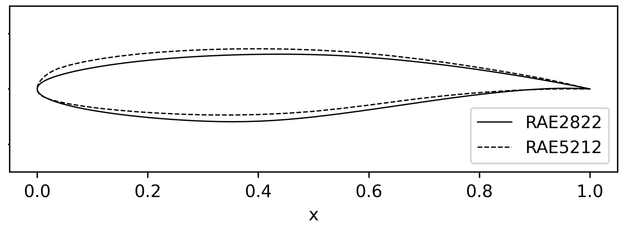
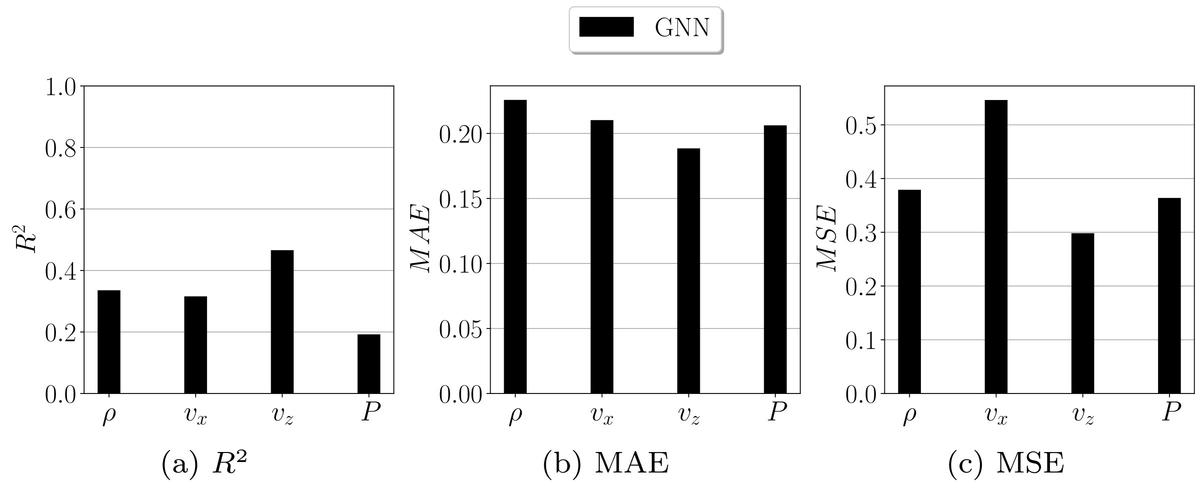


Figure 4.21: RAE2822 vs. RAE5212 airfoil geometry

data. To confirm the lack of extrapolation capabilities, the GNN model trained on the RAE2822 data is now applied to infer a correction for a slightly different geometry, namely the RAE5212 airfoil given in Figure 4.21. To lessen the degree of generalization needs, the same coarse grid as for the RAE2822 is used to generate data, which is simply deformed to accommodate the new RAE5212 airfoil geometry.

To test the GNN model, simulations of the RAE5212 airfoil are conducted at the same flow conditions as for the training airfoil, while keeping the Mach number at $M = 0.6$ and varying the angle of attack between 0.0° and 8.0° . The corresponding high-fidelity simulations on the fine grid are also conducted, but solely for the purpose of comparison with the ML corrected solutions. Figure 4.22a presents the resulting metrics, including coefficient of determination R^2, mean absolut error MAE, and mean squared error MSE, averaged over all RAE5212 simulations. The metrics quantify how well the predicted correction corresponds to the true correction term for each variable, i.e. density ρ, the two velocity coefficients v_x and v_z, and pressure P. Previously, the GNN model has achieved an R^2 value greater than 0.9 for the RAE2822 airfoil. Inferring corrections for the RAE5212 airfoil with the GNN model trained on the RAE2822 airfoil results in a R^2 values between 0.19 for the pressure variable P and 0.46 for the velocity



(a) R^2 (b) MAE (c) MSE

Figure 4.22: Test set metrics for $M = 0.75$, RAE5212

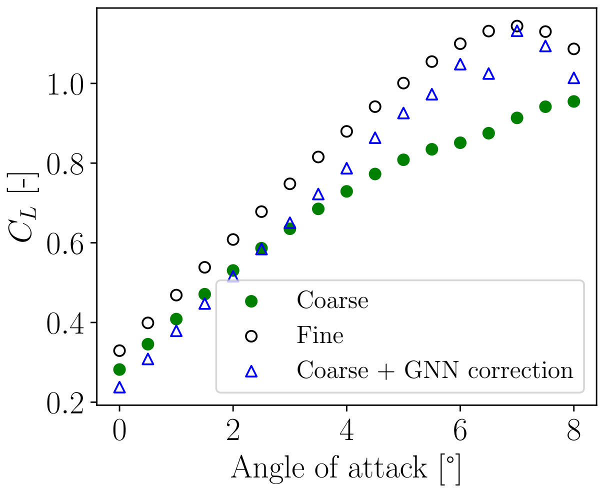
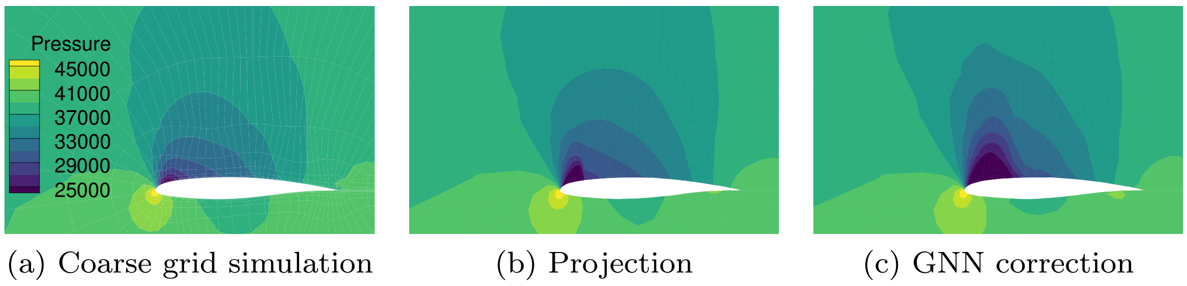


Figure 4.23: Lift coefficient C_L for RAE5212 test set at $M = 0.6$

variable v_z. The MAE for the density variable ρ increases by more than a factor of four, and the MSE for the velocity v_x rises from 0.10 to 0.29.

This decrease of corrective capabilities is also visible when looking at the resulting lift coefficients given in Figure 4.23. As before, the gap between coarse and fine grid simulations is evident. Although the GNN model tends to increase the lift coefficients at high angles of attack towards the high-fidelity solutions, the resulting lift coefficients are significantly poorer for the RAE5212 test set compared to the RAE2822, and even worse for lower angles of attack than the coarse grid simulation.

As a final assessment to display the lack of generalization capabilities to out of distribution data, the pressure field for the test sample at angle of attack $\alpha = 4.0°$ is presented in Figure 4.24. Figure 4.24a shows the coarse grid simulation result and Figure 4.24b the high-fidelity pressure field injected onto the coarse grid. The difference between both is mainly located in front of the shock location. The discontinuity is less pronounced in the solution obtained with the coarse grid. Applying the predicted GNN correction, as given in Figure 4.24c, it can be seen that the correction is overestimated, leading to an exaggerated low pressure area that is larger and more pronounced than expected.



(a) Coarse grid simulation (b) Projection (c) GNN correction

Figure 4.24: Pressure field for $M = 0.6$ and $\alpha = 4.0°$, RAE5212

To improve the generalization capabilities of an ML model, particularly for varying geometries, one approach is to extend the design and training space by incorporating samples of different geometries. This ensures that the model learns from a diverse set of shapes, capturing underlying patterns rather than overfitting to a limited dataset. Additionally, leveraging data augmentation techniques or physics-informed constraints could further enhance robustness, enabling the model to make accurate predictions even for unseen configurations.

Data Preparation and Training Time

A further drawback of the method, akin to any data-driven method, is the time investment required to generate high-fidelity data for the training phase and optimize the hyperparameters of the ML models. Table 4.7 summarizes the number of simulation samples conducted for the RAE2822 and the LANN wing training data, and the resulting simulation time, computed by using the reported average wall clock times given in table 4.1 and 4.6 for both coarse and fine grid simulations, as well as the time needed to train the final GNN model. Obviously, this effort in data collection and model training is only justified if during the inference phase the model is used on a sufficient amount of new simulations to break even the time needed to generate the ML model in the first place. Needing on average 7 hours for a fine grid simulation of the RAE2822, this point would be reached after 61 simulations. For the LANN wing, requiring 5 hours and 10 minutes for one high-fidelity simulation, the break even point would be reached after 43 simulations.

For industrial settings, it can be expected that databases with high-fidelity simulations are already available. Thus, the most costly aspect of the proposed approach, the generation of high-fidelity data, is significantly reduced or even eliminated. Instead of requiring additional expensive computations, these existing datasets can be utilized to develop a surrogate model as presented in this work.

An additional time consuming point which is not accounted for in table 4.7 is the hyperparameter optimization. For the GNN model presented in this chapter, the hyperparameters have been found during Bayersian optimization after the fifteenth iteration, which means that a total of 15 GNN models had to be trained to arrive at the presented results. The number of iterations can vary significantly, and it was found that for the RF model a convergence in improved

Table 4.7: Number of training samples and total time needed for data collection and final GNN model training, FV study

	RAE2822	**LANN**
No. of samples	60	40
Simulation time	421 h 30 min	211 h 47 min
Training time	2 h 12 min	9 h 54 min
Total time investment	423 h 42 min	221 h 41 min

metrics for the validation set is found earlier, while the NN and GNN models tend to need more exploration, which is likely due to their greater number of influential hyperparameters. Nevertheless, it was shown for the 3D LANN wing that expensive hyperparameter optimization can be avoided. For this case, hyperparameters from previous studies, in this work the 2D case, were successfully reused without extensive need of additional tuning.

4.4 Concluding Remarks

This chapter presented a method to quantify the discretization error between FV solutions obtained on two nested grids. After training ML models on collected data obtained from coarse and fine grid simulations, they are employed to infer the error. The predicted error can be used to correct low-fidelity field variables to approximate the corresponding fine grid simulation injected onto the coarse grid. The first aim of this investigation was to assess if the correction term can be successfully learned. For this, three different models were trained on the 2D test case, taking as inputs the first and second derivatives of the variables to be corrected and a local cell Reynolds number. All models show corrective capabilities on the test set. Difficulties are generally encountered in the flow regime at high angles of attack, where discontinuities are dominant. As for the training, the RF stands out for its simplicity: relatively few hyperparameters are quickly tuned to reach convergence on the validation metric, whereas both neural network-based models require more extensive hyperparameter optimization, and often longer training times to achieve similar or superior performance. Judged only by their corrective capabilities, the GNN model outperforms both NN and RF, as the latter two show noisy behavior in their corrections. The overall better performance underlines the GNN's capability to grasp more information by taking into account the connectivity to neighboring nodes and their features, resulting in overall smoother field corrections and less oscillatory surface values such as the pressure coefficient. This is shown not only on a 2D test case, but also a 3D test case, involving turbulent flows.

These findings indicate that the discretization error with respect to a fine grid simulation can be quantified, learned and finally corrected. Nevertheless, limitations of the proposed method need to be highlighted as well. These include the degradation of the fine grid simulation after projecting it onto the low-fidelity discretization, i.e. the coarse grid. For the cases investigated, coarse boundary layers lead to insufficient accuracy of velocity gradients, ultimately limiting the accuracy of values such as the skin friction coefficient and thus the friction component of the resulting drag. From the ML side, limitations are posed by the generalization capabilities of the ML model, showcased by applying the GNN model trained on an RAE2822 airfoil to data of an RAE5212 airfoil. Although the corrections tend towards the right direction, such as increasing the resulting lift coefficient or amplifying the shock, the accuracy is not comparable to results obtained for the RAE2822 test data. Further limitation posed by the data-driven approach include the time investment needed to collect training data and find optimal hyperparameters. These issues can most likely be avoided in industrial settings and with sufficient experience in ML model training.

Two practical restrictions of the proposed approach to correct FV simulations are the need to collect data on two grids and defining a mapping function to transfer the fine grid solution to the coarse grid discretization. In the following chapter, the method is adapted to correct low-order polynomial DG solutions instead of coarse grid simulations. With this, the need of having multiple grids to generate low- and high-fidelity solutions is eliminated, since solutions of different accuracies can be obtained on the same grid. Additionally, the choice of mapping function becomes trivial.

Chapter 5

Application: Correction of Steady DG Simulations

Discontinuous Galerkin methods allow to raise the number of degrees of freedom not only by grid refinement, but also by increasing the polynomial degree. Therefore, on the same grid, varying accuracies can be achieved by changing the polynomial degree. In this chapter, advantage is taken of this fact such that both low- and high-fidelity simulations are obtained on the same mesh. An exemplary simulation conducted on the same mesh but across varying polynomial degrees is illustrated in Figure 5.1 for laminar flow around the NACA0012 geometry. The figure shows that with increased number of degrees of freedom the lift coefficient becomes more accurate, until it approximates a reference value. At the same time, the computational cost, here quantified by the wall clock time needed to run the simulations, also rises. Neither accuracy nor cost grow linearly with the

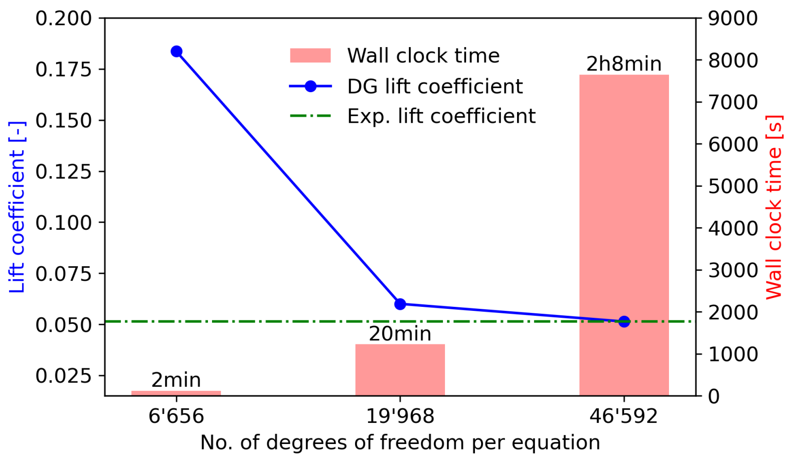


Figure 5.1: Simulation time vs. accuracy on varying polynomial degrees

number of degrees of freedom, resulting in high computational cost for accurate solutions. As a consequence, conducting simulations with the DG discretization suffers from the same trade-off between accuracy and computational cost as the FV discretization.

In the previous chapter, the high-fidelity simulations are conducted using the FV discretization on fine grids. Projecting this solution onto coarser grids leads to a loss of accuracy. Furthermore, the choice of mapping function for this projection is not straightforward. Deploying a modal DG discretization, only a single coarse grid can be employed, while both low- and high-fidelity simulations are computed by varying the polynomial degree. Additionally, the choice of a mapping function is simple. This, since a truncation of the higher polynomial terms leads to an L_2 projection, as the polynomial basis used in this work is orthogonal, as described in section 3.1.2. The results of this approach are investigated in the following on two test cases, namely on the turbulent flow around the 2D RAE2822 airfoil and on the laminar flow around a 3D delta wing. The goal is to infer an ML prediction which can be used to correct inaccurate solutions obtained with low polynomial degrees. The content of this chapter has partially been published in [85].

5.1 2D Case - RAE2822 Airfoil

The first test case under consideration is the same as for the FV studied in section 4.1, that is the RAE2822 airfoil. For the dataset generation, the Reynolds number is kept constant at $Re = 5.7 \times 10^6$, and the same design of experiment is conducted as for the FV study.

5.1.1 Data Generation

In the following, the simulation setup to generate a dataset is described. For one sample, the different results are showcased and compared. Here, the main focus is to present the difference between solutions of lower and higher polynomial degree, and to proof that the high-order solution truncated to the low-fidelity discretization is of higher accuracy than the low-order one.

Simulation Setup

For the DG simulations, the RANS equations are solved with the negative version of the SA turbulence model. For both high and low-order simulations, the same C-type grid is employed with 2'464 quadratic elements, given in Figure 5.2. Since this case features discontinuities at high angles of attack, which has shown to corrupt the robustness for certain samples, a shock sensor and artificial viscosity are used for polynomial degree $p > 0$. Convergence is reached for all simulations, defined by a reduction of 12 orders of magnitude for all residuals, including the turbulent variable $\tilde{\nu}$. Simulations are conducted for polynomial degrees of $p \in \{0, 1, 2\}$. The lower degrees of $p = 0$ and $p = 1$ are considered to be corrected, whereas $p = 2$ serves as high-fidelity solution to be approximated. For degrees $p = 1$

Table 5.1: Number of elements, degrees of freedom per equation, and average wall clock time for DG study, RAE2822

Polynomial degree	**Elements**	**Degrees of freedom**	**Time**
$p = 0$	2’464	2’464	30 s
$p = 1$	2’464	7’392	5 min
$p = 2$	2’464	17’248	50 min

and $p = 2$, the solution obtained on the previous polynomial degree is used as restart solution to accelerate convergence. The resulting number of degrees of freedom and the average wall clock time per simulation are reported in table 5.1. The table highlights the significant increase in degrees of freedom when choosing higher order polynomials, as well as higher computational time needed to conduct these simulations. As for the FV study on the RAE2822, low- and high-fidelity simulations are run for all sample points. The design space and sampling strategy are the same as for the FV study, including 60 points for training and additional 29 simulations for testing, as presented in section 4.1.1 and given in Figure 4.4. Although this case features the same sample points as for the FV study, both studies employ different grids and thus different numbers of degrees of freedom, as the DG discretization requires for this case quadratic elements. The main goal is not to compare the results on both FV and DG, but rather show the versatility of the correction approach on different discretizations.

Simulation Results

Figure 5.3 presents the pressure coefficient for the simulation at Mach number $M = 0.75$ and angle of attack $\alpha = 5.5°$. Comparing the simulation results obtained with $p \in \{0, 1, 2\}$, it is visible that the lowest order does not develop a shock similar to the one seen at higher orders. For $p = 0$, the pressure coefficient along the suction side deviates significantly from the $p = 2$ solution, while less pronounced deviations are present along the pressure side of the airfoil. For the pressure coefficient obtained with polynomial degree $p = 1$, the difference to the

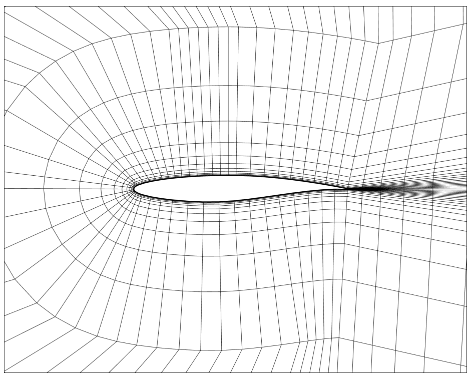


Figure 5.2: Grid used for the RAE2822 DG study

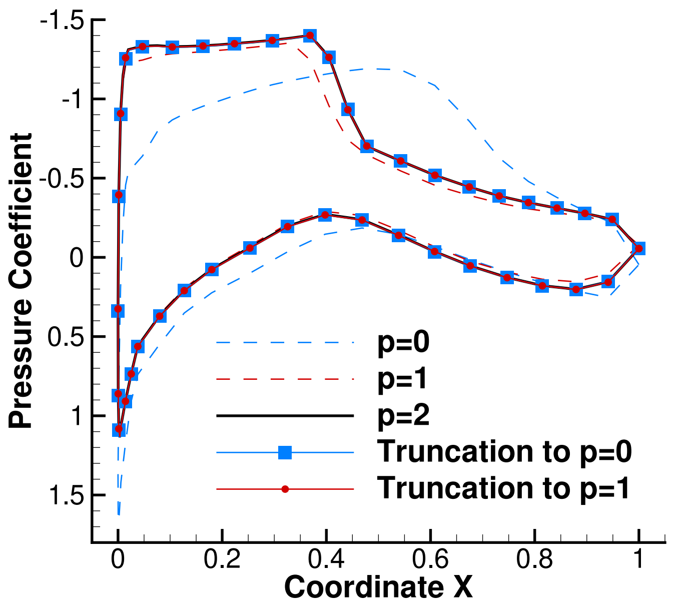


Figure 5.3: Pressure coefficient C_P for $M = 0.75$ and $\alpha = 5.5°$ obtained with low and high-order polynomial degree $p \in \{0, 1, 2\}$, as well as C_P from truncated high-order solution

higher order $p = 2$ is reduced, but still present especially around the shock location. Furthermore, Figure 5.3 shows the derived pressure coefficient from the high-order solution after the variables of interest have been truncated to the low-fidelity discretization space. It is evident that the pressure coefficient obtained on the high-fidelity discretization is preserved, for both $p = 0$ and $p = 1$ discretizations. Thus, this indicates that the low-order solution can be corrected to potentially achieve this level of accuracy.

Additionally, the simulation results are compared with available measurement and reference data from a test campaign [1] and simulations from literature [105]. For this, CODA simulations are conducted at $M = 0.676$ and $\alpha = 1.93°$. Table 5.2 depicts the lift coefficient of these sources, the lift coefficient derived from simulations with varying polynomial degree, and the resulting lift count with respect to

Table 5.2: Lift coefficient C_L for simulation with $p \in \{0, 1, 2\}$, reference values and test campaign, at $M = 0.676$ and angle of attack $\alpha = 1.93°$, RAE2822

Polynomial degree / reference	C_L	**Lift count difference**
$p = 0$	0.436	130
$p = 1$	0.505	61
$p = 2$	0.570	4
Ref. 1 [105]	0.569	3
Ref. 2 [105]	0.561	5
Test campaign [1]	0.566	-

the test campaign source. The table highlights the inaccuracy of the simulations with low polynomial degrees $p \in \{0, 1\}$, as the lift count is significantly higher than for the solution with $p = 2$ and the reference values. Thus, the simulations obtained with $p = 2$ and the corresponding simulation settings are deemed accurate enough for the following study.

5.1.2 Machine Learning Correction

The ML model training follows a similar procedure to the one discussed in the previous chapter, presented in section 4.1.2, but is nevertheless shortly described in the following. For this case, only two models are employed, and their resulting correction capabilities are presented on the test set samples for both polynomial degrees $p = 0$ and $p = 1$.

Training Setup

To correct DG solutions, the variables of interest are indirectly corrected by the ML prediction, by adding the corrective term to the degrees of freedom $\boldsymbol{a}$ and not directly to the variables $\tilde{\boldsymbol{u}}$ themselves, as described in 3.2.2. The variables of interest include the conservative variables of density ρ, momentum M_x and M_z, as well as energy ρE, such that $\tilde{\boldsymbol{u}} = [\rho, \boldsymbol{M}, \rho E]^T$. Note that as for the FV study, the turbulent variable $\tilde{\nu}$ is not considered for correction. Thus, for polynomial degree $p = 0$, there are 4 correction terms to be predicted, while the model receives 25 inputs per element. These inputs include, given in (3.16) the first and second derivative of the degrees of freedom, computed by a weighted least squares approach, and a local cell Reynolds number, given in (3.14). There are no significant differences compared to the FV approach for $p = 0$, since the first basis function is $\phi = 1$. For the correction of $p = 1$ solutions, there are 3 degrees of freedom to be corrected per equation for a 2D case, resulting in 12 predictions per element. The feature inputs scale accordingly, resulting in 73 values per element. As for the choice of ML models, only the RF and GNN are considered in this study. The RF is chosen for its ease of training, making it an ideal baseline model, while the GNN is selected as it has previously demonstrated superior performance compared to both RF and NN. To find optimal hyperparameters, a four-fold cross validation is performed with Bayesian optimization for the $p = 0$ dataset. For the RF, the hyperparameters to be tuned include the number of decision trees, the number of features to be considered for a split, and if bootstrapping is applied or

Table 5.3: RF hyperparameters, RAE2822

Hyperparameter	**Value**
No. of decision trees	428
Bootstrapping	False
Features per split	Square root criterion

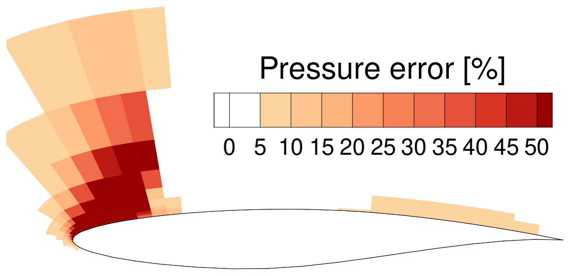
Table 5.4: GNN hyperparameters, RAE2822

Hyperparameter	**Value**
No. of layers	9
Feature dimension per layer	298
Initial learning rate ($p = 0$)	1.955×10^-3
Initial learning rate ($p = 1$)	1.973×10^-3

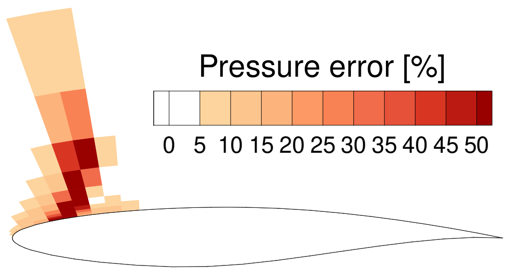
not. The final hyperparameters are reported in table 5.3. The number of features per split is determined by the square root of the maximum number of available features. For the GNN, residual gated graph network layers were found to perform better than GCN layers, and further hyperparameters are reported in table 5.4. Some hyperparameters are kept based on the FV study: such as an exponential learning rate decay and hyperbolic tangent as non-linear activation function. For the $p = 1$ models, the same architectures are employed as for $p = 0$, but the initial learning rate is adjusted for the GNN model based on another four-fold cross validation. The final RF is trained on CPU with 6 parallel jobs and requires for $p = 0$ around 5 minutes and for $p = 1$ around 15 minutes. For the training of the GNN model, a NVIDIA Quadro P2200 GPU with 5 GB memory is employed, resulting in training times of approximately 45 minutes and 1 hour for $p = 0$ and $p = 1$, respectively.

Correction Results

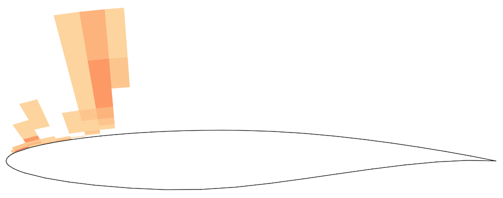
The first qualitative results of interest are the corrected fields. Figure 5.4 depicts the percentage error of the pressure field for the test sample at Mach $M = 0.6$ and angle of attack $\alpha = 7.5°$. Thus, not the corrected degrees of freedom nor any of the variables $\tilde{\boldsymbol{u}}$ are shown, but rather the derived pressure field to be consistent with the results presented in the previous chapter. The two figures on the top, namely Figure 5.17a and Figure 5.17b, show the percentage error between the truncated high-fidelity solution and the low-order solution for $p = 0$ and $p = 1$, respectively. The error of the low-fidelity discretization is mainly located in front of the shock location, and it is visible that the lower the order the greater the discrepancies become. For $p = 0$, additional errors are visible towards the trailing edge. The subsequent rows show the percentage error of the resulting pressure field after the ML corrections. Figure 5.17c and Figure 5.4d present the fields corrected by the RF model for $p = 0$ and $p = 1$, while Figure 5.4e and Figure 5.4f depict the respective GNN corrected fields. At a first glance, the GNN corrections feature lower errors around the shock location. Nevertheless, when assessing the error close to the boundary, the GNN corrected field exhibits greater errors in close vicinity of the airfoil. Overall, the percentage error for the pressure field is reduced from 172.9% to 42.7% with the RF and to 63.0% with the GNN for $p = 0$, and from 96.8% to 35.8% with the RF and to 32.9% with the GNN for $p = 1$. For the GNN corrections, the maximum error is located within the boundary layer,



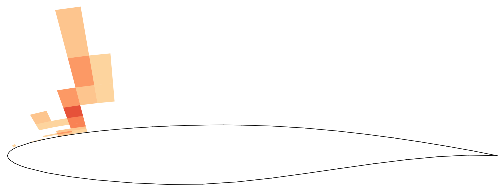
(a) $p = 0$ simulation



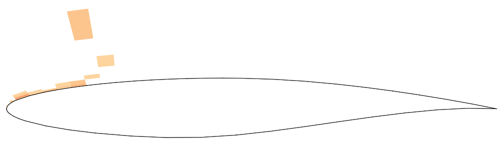
(b) $p = 1$ simulation



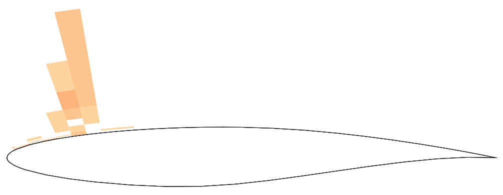
(c) $p = 0$ with RF correction



(d) $p = 1$ with RF correction



(e) $p = 0$ with GNN correction

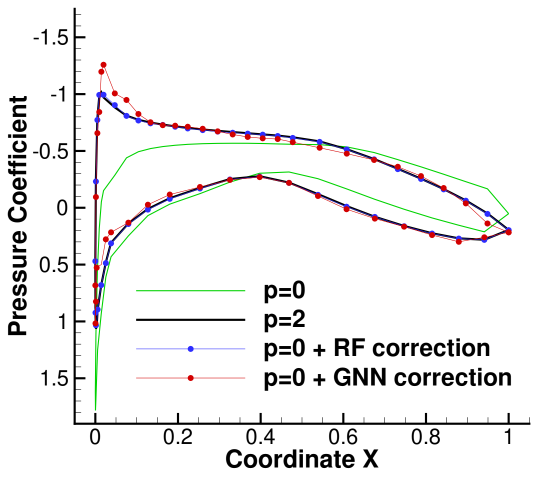


(f) $p = 1$ with GNN correction

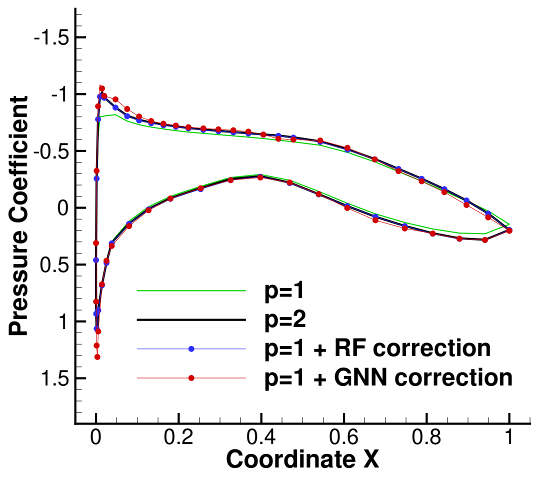
Figure 5.4: Pressure error [%] for $M = 0.6$ and $\alpha = 7.5°$ compared to the truncated high-fidelity simulation, RAE2822

which is not the case for the RF results.

The different corrective capabilities close to the airfoil boundary is also reflected when evaluating surface related values, such as the pressure coefficient. Figure 5.5 shows the pressure coefficient for a test sample at lower Mach number $M = 0.6$ and angle of attack $\alpha = 2.0°$ on the left for $p = 0$ in Figure 5.5a, and on the right for $p = 1$ in Figure 5.5b. Similarly, the pressure coefficient is depicted for a higher Mach number of $M = 0.75$ and angle of attack of $\alpha = 5.0°$ in Figure 5.6. It is again obvious that the $p = 0$ solution differs more significantly from the high-fidelity simulation. At lower Mach number given in Figure 5.5a, the RF correction perfectly matches the high-fidelity pressure coefficient, while the GNN correction exhibits slight deviations, resulting in noisy behavior especially for $p = 0$. Assessing the case with higher Mach number and higher angle of attack, where a shock is present, the RF starts to deviate from the high-order solution, especially around the discontinuity. As in the FV study, the GNN shows improved corrective capabilities for discontinuities. Nevertheless, the GNN still shows noisy

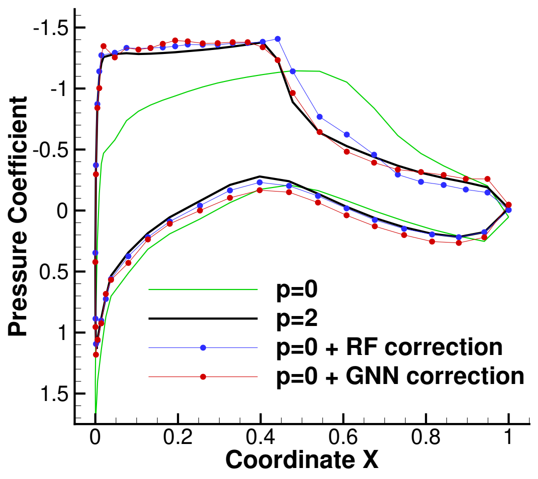


(a) $p = 0$

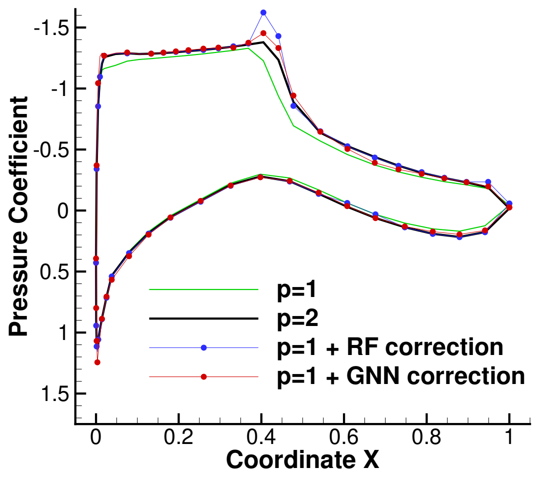


(b) $p = 1$

Figure 5.5: Pressure coefficient C_P for $M = 0.6$ and $\alpha = 2.0^\circ$ for high-order, low-order, and ML corrected solutions, RAE2822



(a) $p = 0$



(b) $p = 1$

Figure 5.6: Pressure coefficient C_P for $M = 0.75$ and $\alpha = 5.0^\circ$ for high-order, low-order, and ML corrected solutions, RAE2822

tendencies. For both models, the $p = 1$ corrections match the high-fidelity one better than the $p = 0$ corrections.

Figure 5.7 and Figure 5.8 show the derived lift coefficients from low and high-order solutions, as well as from the ML corrected solutions for both test sets, for polynomial degree $p = 0$ and $p = 1$, respectively. Both ML models achieve in general good performance at lower angles of attack. For the $p = 0$ corrections, this performance degenerates at higher angles of attack, mirroring the trends observed in the previous FV study, highlighting the similarity between FV and DG discretization with $p = 0$. The uncorrected low-order solution shows a linear

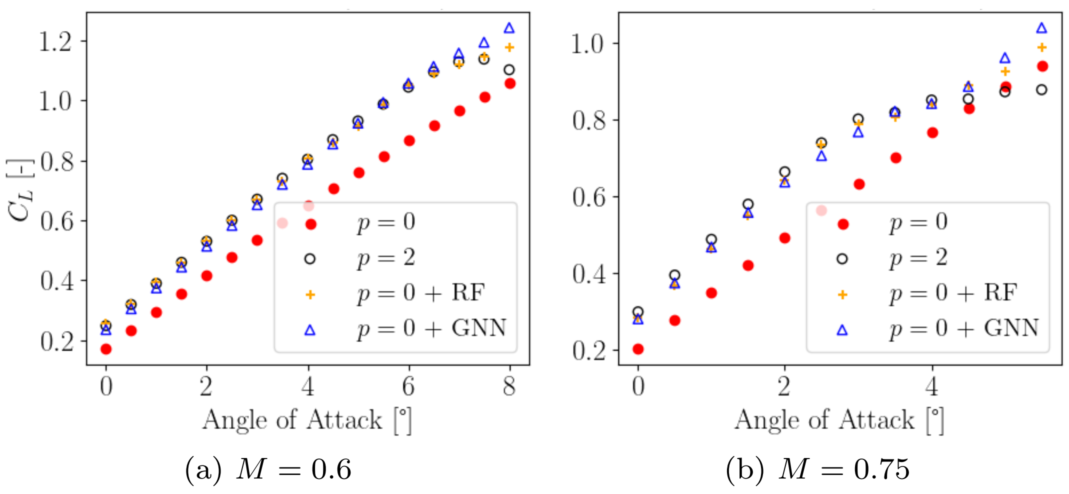


Figure 5.7: Lift coefficient C_L for test sets and polynomial degree $p = 0$, RAE2822

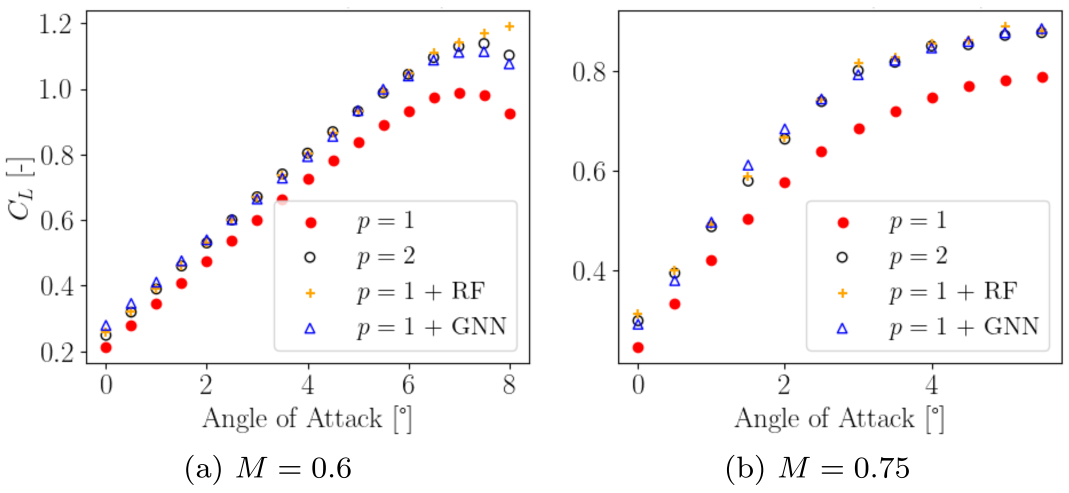


Figure 5.8: Lift coefficient C_L for test sets and polynomial degree $p = 1$, RAE2822

trend for the lift coefficient, which suggests that the flow field might lack sufficient complexity, leading to feature inputs that are not rich enough for the ML models to effectively learn meaningful corrections at high angles of attack. Examining the lift coefficient derived from the $p = 1$ simulation in Figure 5.8, it is visible that the low-order simulation already exhibits a non-linear behavior but nevertheless deviates from the high-fidelity results, especially at high angles of attack which is even more pronounced for the higher Mach number. Both ML model corrections improve the low-fidelity lift coefficient, with the GNN showing superior corrective capabilities for Mach $M = 0.6$ and higher angles of attack. At lower angles of attack, the RF corrections match the high-order solutions better, while the lift coefficients derived from the GNN corrections show noisier behavior. From this

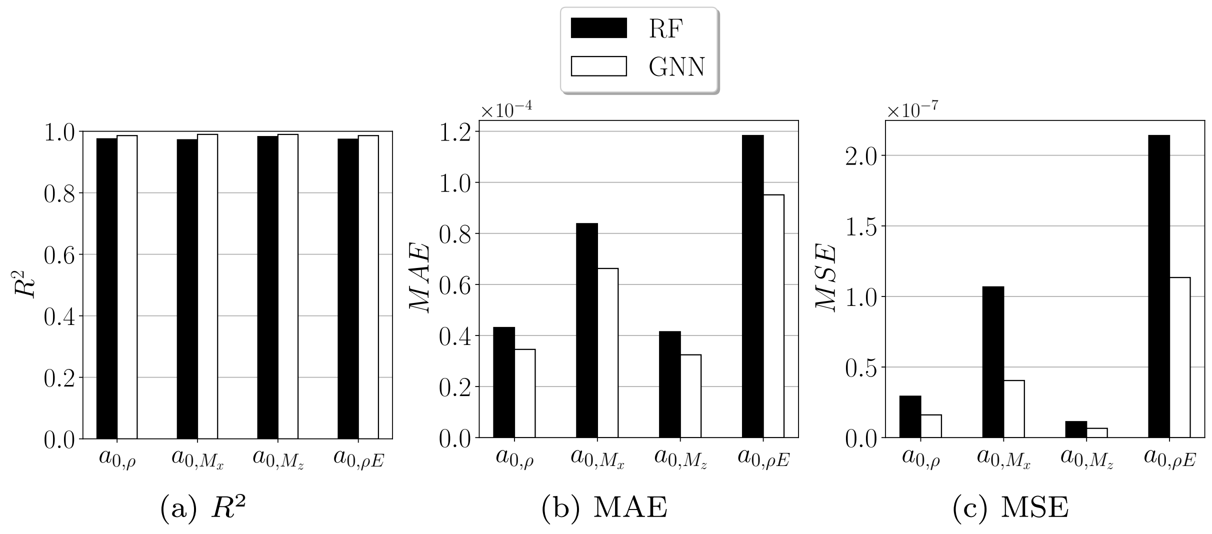


Figure 5.9: Test set metrics for $p = 0$ and $M = 0.6$, RAE2822

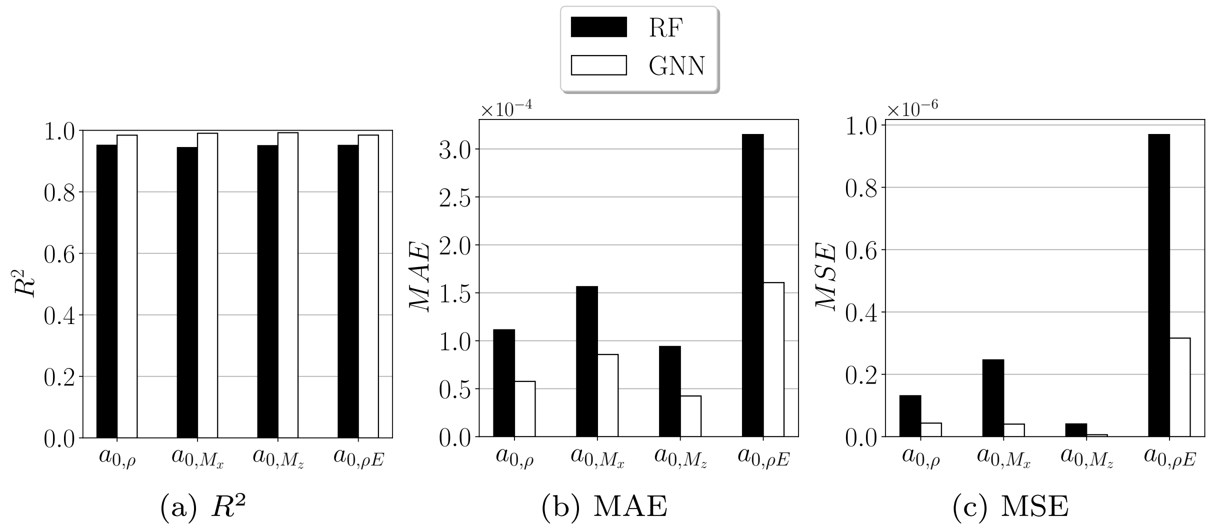


Figure 5.10: Test set metrics for $p = 0$ and $M = 0.75$, RAE2822

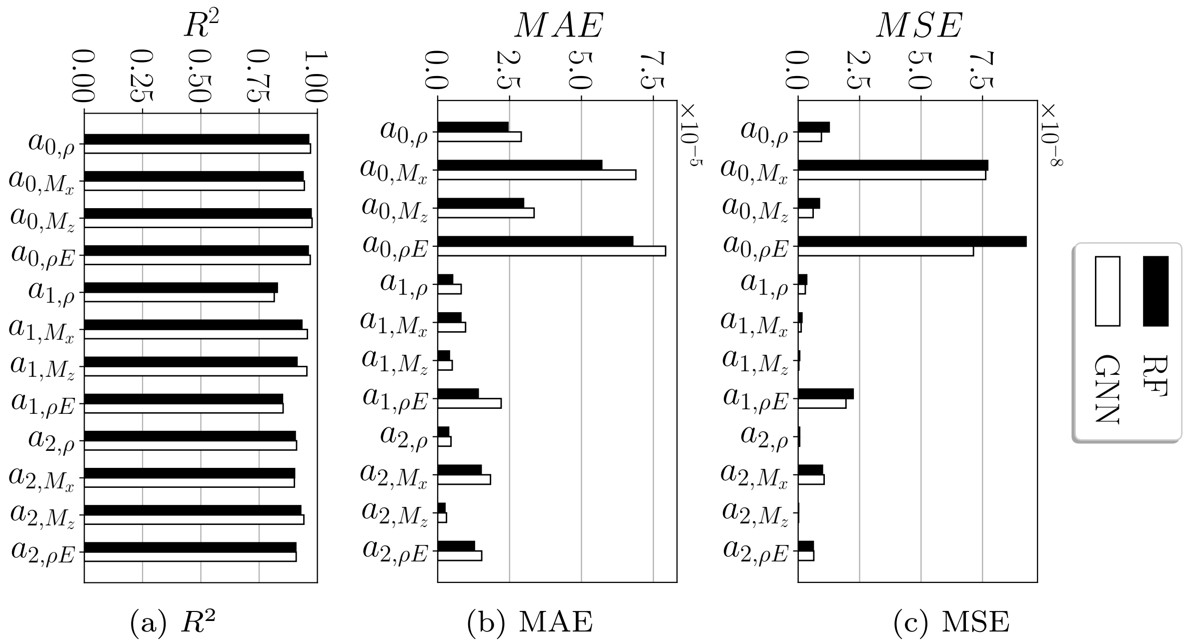
it can be again concluded that the RF correction results in overall better surface related values when shocks are not present, since its predictions at the boundary layer elements is better and smoother compared to the ones from the GNN model.

Finally, to assess the overall corrective capabilities of both ML models quantitatively, the coefficient of determination R^2 (2.41), the MAE (2.40), and the MSE (2.39), are given in Figures 5.9 and 5.10 for the $p = 0$ test set simulations and in Figures 5.11 and 5.12 for the $p = 1$ test set simulations. Here, the metrics are computed for the correction of each polynomial coefficient and evaluated across all test instances. For the $p = 0$ predictions, both RF and GNN model show high R^2 values, with the GNN exhibiting slightly better performance. For the test samples with Mach number $M = 0.6$, the lowest score for the RF is 0.972 for

a_{0,M_x}, i.e. the first polynomial coefficient of the momentum variable, while the lowest score of 0.986 for the GNN is found for $a_{0,\rho E}$. For the test set at Mach number $M = 0.75$, the RF has its lowest score with $R^2 = 0.944$ for a_{0,M_x}, while the GNN reaches the lowest score of $R^2 = 0.984$ for $a_{0,\rho}$. Comparing the values between RF and GNN indicate that the GNN returns slightly better predictions across the whole flow field than the RF, which is also visible when looking at the MAE and MSE. However, the previous evaluations of pressure and lift coefficient values suggest that these metrics alone are not a sufficient indicator to assess the models overall performance, if the main interest lies in surface related values.

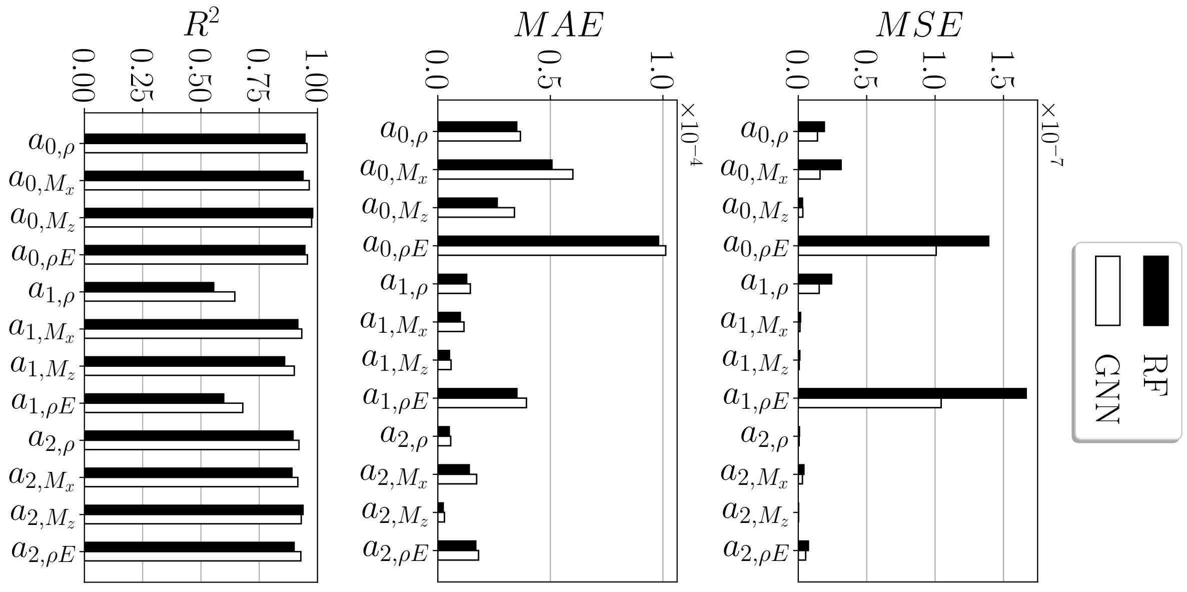
In Figures 5.11 and 5.12 the metrics are given for both test sets for the $p = 1$ corrections, underlining the increased number of degrees of freedom when rising the polynomial degree only by one, resulting in three times as many predictions. Again, there is a trend for the GNN to achieve higher R^2 values than the RF, although this is not the case for all polynomial coefficients. In general, the second degrees of freedom $\boldsymbol{a}_1$ exhibit a performance drop, especially for the density and energy variable. Overall, the lowest value for Mach number $M = 0.6$ is for both models found for the polynomial coefficient $a_{1,\rho}$, resulting in $R^2 = 0.828$ for the RF, and a slightly lower value of $R^2 = 0.815$ for the GNN. It is also again visible that the RF achieves in general lower MAE values for Mach number $M = 0.6$. Nevertheless, the MAE shows here for both models in general low values.

For higher Mach number $M = 0.75$, again both models exhibit the lowest coefficient of determination for $a_{1,\rho}$, with $R^2 = 0.555$ for the RF and $R^2 = 0.650$ for the GNN. This indicates again better corrective capabilities for the GNN in cases with non-linearities. The MSE values also support this, showing



(a) R^2 (b) MAE (c) MSE

Figure 5.11: Test set metrics for $p = 1$ and $M = 0.6$, RAE2822



(a) R^2 (b) MAE (c) MSE

Figure 5.12: Test set metrics for $p = 1$ and $M = 0.75$, RAE2822

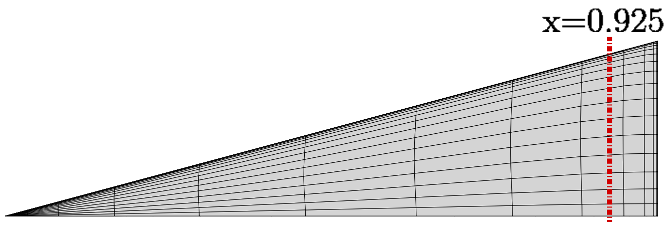
lower values for the GNN than for the RF. The MAE is lower for the RF. Spikes across all metrics are visible for $a_{1,\rho E}$, the second polynomial coefficient for energy, highlighting general difficulties for the correction of this specific variable.

5.2 3D Case - Delta Wing

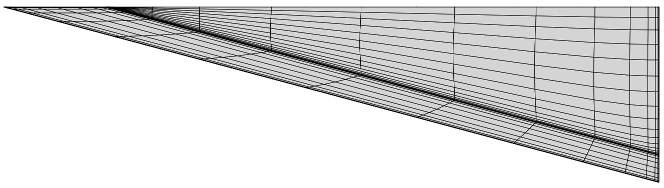
To scale the proposed method, in this section both RF and GNN are trained to infer the correction of low-order simulations of a 3D delta wing. For this, laminar flow is considered with constant Reynolds number $Re = 4'000$. The trailing edge of the delta wing geometry is blunt and the leading edge sharp and sloped. At higher angles of attack, vortices are formed as the flow passes the leading edge. In [112], multiple high-order DG simulations have been carried out on a series of grids at flight conditions $M_{inf} = 0.3$ and angle of attack $\alpha = 12.5°$, a test case defined by a EU project called ADIGMA [97], which will be used as reference condition.

5.2.1 Data Generation

In the following, the simulation setup is described, with which samples with varying polynomial degree are computed within a design space which is spanned by different Mach numbers and angles of attack. The simulation results are compared at $M_{inf} = 0.3$ and angle of attack $\alpha = 12.5°$, a flight condition proposed by [97]. For this case, a reference lift coefficient is available from [112], with which the simulation settings and accuracy of the high-order simulations can be confirmed.



(a) Upper wing surface



(b) Lower wing surface

Figure 5.13: Grid used for the delta wing

Simulation Setup

For all simulations, the Navier-Stokes equations are solved. The same mesh with 26'356 linear elements is used for all polynomial degrees. So far, the previous application cases were mainly focusing on surface and integral values. Employing for this test case a grid with relatively many elements, in this case 26'356 compared to the 8'236 in the 3D FV case, allows to resolve and correct for phenomena of interest appearing in the flow field, which in this case are the vortices being formed by the flow passing the wing edge. Employing linear elements are sufficient, since the geometry does not show any curvature. The mesh of the upper and lower wing surface is presented in Figure 5.13. No artificial viscosity is added, since no significant discontinuities are anticipated in the chosen design space. As for the previous 2D case, simulations are conducted for polynomial degrees $p \in \{0, 1, 2\}$, with the aim to correct the $p \in \{0, 1\}$ solutions to approximate the highest order solution obtained with $p = 2$. As the number of degrees of freedom scales with polynomial degree p and dimensionality of the problem d, as described in (2.17), this number rises significantly from $p = 0$ to $p = 2$, as given in table 5.5. The table also shows that the high-order solution requires much more time, resulting in an average wall clock time of over 7 hours for $p = 2$, while on average the lower order simulations are conducted within minutes.

Table 5.5: Number of elements, degrees of freedom per equation, and average wall clock time, delta wing

Polynomial degree	**Elements**	**Degrees of freedom**	**Time**
$p = 0$	26'356	26'356	1 min
$p = 1$	26'356	105'424	17 min
$p = 2$	26'356	263'560	7 hrs 40 min

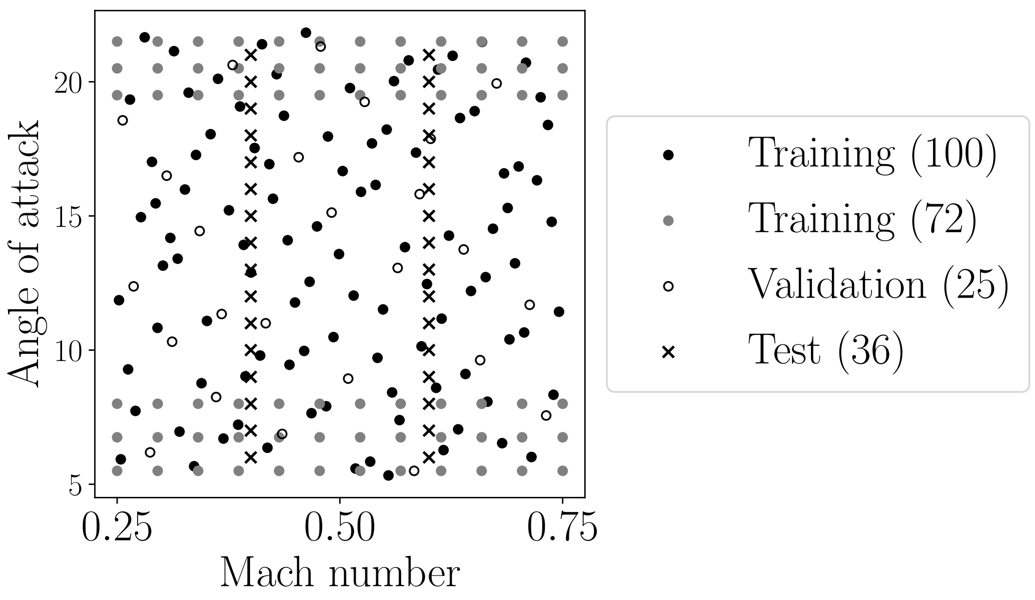
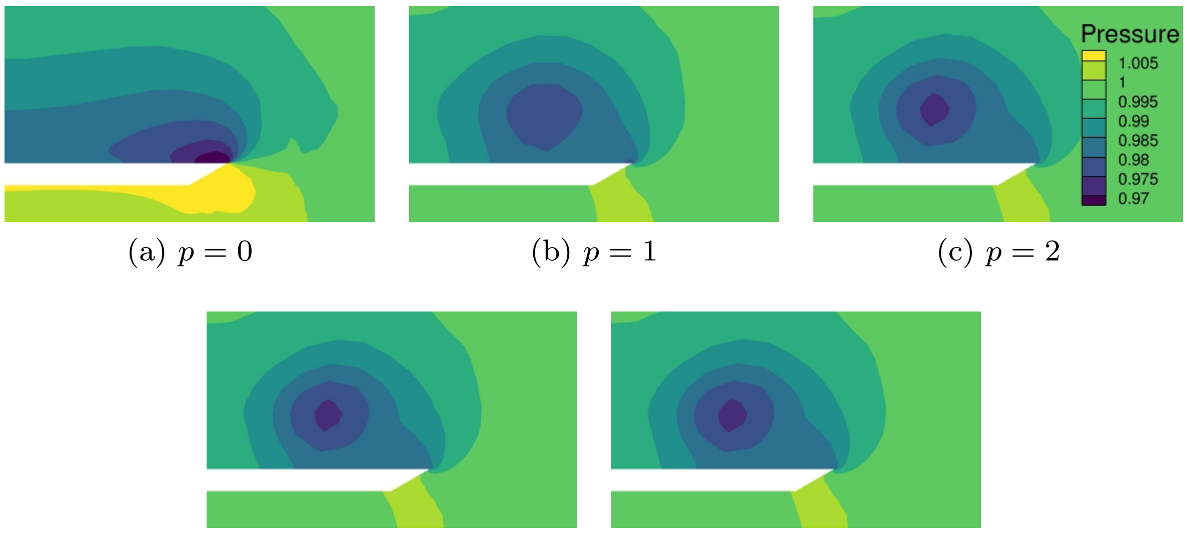


Figure 5.14: Sampling points for train, validation and test sets, delta wing

For all polynomial degrees, simulations are computed at each sample point given in Figure 5.14. The design space is spanned by angle of attack between $6.0° \leq \alpha \leq 21.0°$ and Mach number between $0.25 \leq M \leq 0.75$. Since vortices are formed at higher angles of attack, values of $\alpha \leq 5.0°$ have been removed deliberately. Initially, a Halton sequence is used to generate 125 samples, out of which 100 are used for the ML model training and 25 as hold out validation samples. Early ML training results on the validation set have shown that the predictive capabilities decrease at high and low angles of attack. Thus, additional 72 points are added using a full factorial sampling to increase the data density in these areas, as indicated by the gray points in Figure 5.14. Finally, simulations are conducted at Mach numbers $M = 0.4$ and $M = 0.6$, while sweeping the angles of attack between $6.0° \leq \alpha \leq 21.0°$. These samples act as test set to judge the final ML model corrections.

Simulation Results

To assess the accuracy of the simulations, the reference point proposed by project ADIGMA at $M = 0.3$ and $\alpha = 12.5°$ is selected [97]. Since the flow passing by the delta wing creates a vortex, it is of interest to assess the simulations in this regard. For this, Figure 5.15 displays the pressure field along a cut in the x-axis, namely at $x = 0.925$ close to the trailing edge, for which the cut location is shown in Figure 5.13. The simulation result obtained with highest polynomial degree $p = 2$ shows a shedded vortex, with a relatively low pressure at its core, given in Figure 5.15c. The $p = 1$ result in Figure 5.15b looks similar, but with higher pressure at its center. For $p = 0$, given in Figure 5.15a, no shedding is visible. Figures 5.15d and 5.15e in the bottom display the derived pressure field, after the $p = 2$ degrees of freedom for the conservative variables have been truncated to



(a) $p = 0$ (b) $p = 1$ (c) $p = 2$

(d) Truncation to $p = 0$ (e) Truncation of $p = 1$

Figure 5.15: Pressure field along cut at $x = 0.925$ at $M = 0.3$ and $\alpha = 12.5°$, delta wing

the respective lower order discretization. It is visible that the vortex core retains its strength and that the pressure field is quantitatively approximating the high-fidelity solution. Thus, this suggests that a ML correction might possibly correct the flow field to this extent.

To further assess if the high-fidelity solution obtained with $p = 2$ is indeed accurate, the resulting lift coefficient C_L is compared with the reference value from [112], as presented in table 5.6, showing the respective lift count. The high lift counts for lower polynomial degrees, especially for $p = 0$, highlight the inaccuracy of these simulations, while the higher order simulation with $p = 2$ is comparably close to the reference value. Thus, for this application case, it is a reasonable choice to correct simulations of polynomial degree $p \in \{0, 1\}$ to approximate solutions of $p = 2$.

Table 5.6: Lift coefficient C_L for $p \in \{0, 1, 2\}$ and reference value at $M = 0.3$ and $\alpha = 12.5°$, delta wing

Polynomial degree / reference	C_L	Lift counts
$p = 0$	0.612	265
$p = 1$	0.366	19
$p = 2$	0.352	5
Ref. [112]	0.347	-

5.2.2 Machine Learning Correction

The training of the ML models is based on the findings of the 2D case and described in the following, whereas test set results are only presented for the model which performs best on the validation data.

Training Setup

For the FV study, the GNN outperformed the other ML models and scaling it to the 3D case was done by simply adapting the number of epochs. For the current case, both models, the RF and the GNN, are evaluated, as the GNN did not significantly outperform the RF model during the 2D DG case. Additionally, it was shown that the GNN mainly achieves better performance for cases exhibiting significant discontinuities or in areas where the lift coefficient behaves non-linear, which is not encountered for the delta wing data set. The training is conducted on the 172 training samples and validated on the 25 hold-out validation samples. To learn the correction for $p = 0$, the same architectures are employed which were found for the 2D case, while only adapting during a Bayesian optimization the learning rate and the number of decision trees for the GNN and the RF, respectively. The initial learning rate found for the GNN resulted in a value of 0.00139 and the number of decision trees reduced to 208. The resulting validation metrics are reported in table 5.7. Both models achieve relatively high R^2 and low MSE. Nonetheless, as both these metrics indicate better predictive capabilities for the RF, only this model is applied for the $p = 1$ training and the following results present only the RF corrected simulations.

The RF training for $p = 0$ takes 9 hours and 25 minutes on CPU with 6 parallel jobs. Compared to previous cases, also run on the same amount of parallel jobs, this is a significant increase in training time, but reasonable due to increased number of decision trees and the higher amount of training data: the training set consists of 172 simulations with each simulation having 26'356 degrees of freedom for $p = 0$, resulting in 4'533'232 training instances. Obviously, by increasing the number of parallel jobs, the training could be accelerated. As an element-wise correction is learned, the number of training instance is the same for the $p = 1$ training, but for each training instance there are significantly more input features and four times more outputs to be predicted. With the variables of interest to be corrected being $\tilde{\boldsymbol{u}} = [\rho, M_x, M_y, M_z, \rho E]^T$ and four polynomial coefficients per variable, there are in total 20 outputs to be predicted and 241 input features per element for $p = 1$. In contrast, for $p = 0$, there are 5 outputs and only 61 input features. On the same hardware with the same amount of parallel jobs, the RF

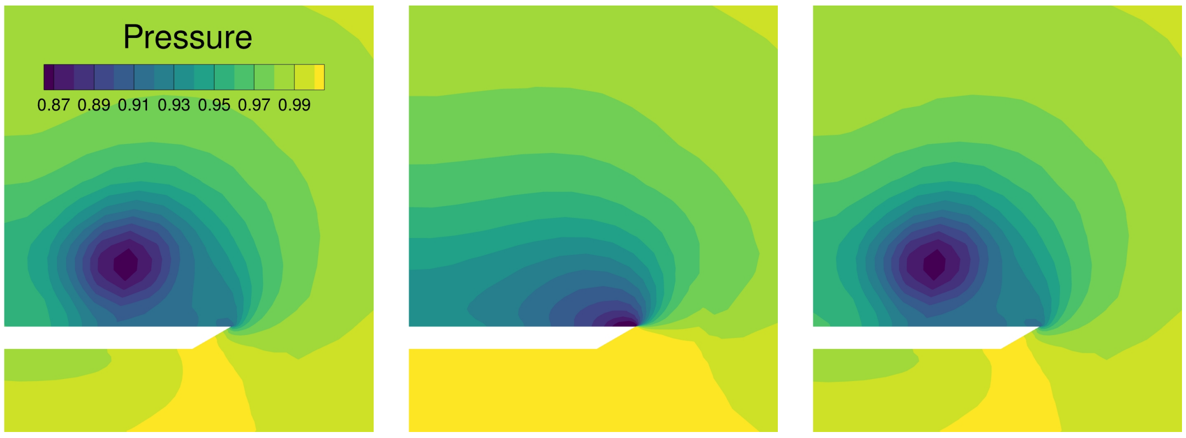
Table 5.7: $p = 0$ validation metrics, delta wing

	$\boldsymbol{R^2}$	**MAE**
RF	0.998	1.23×10^{-6}
GNN	0.966	8.64×10^{-5}

training for the $p = 1$ correction takes 47 hours.

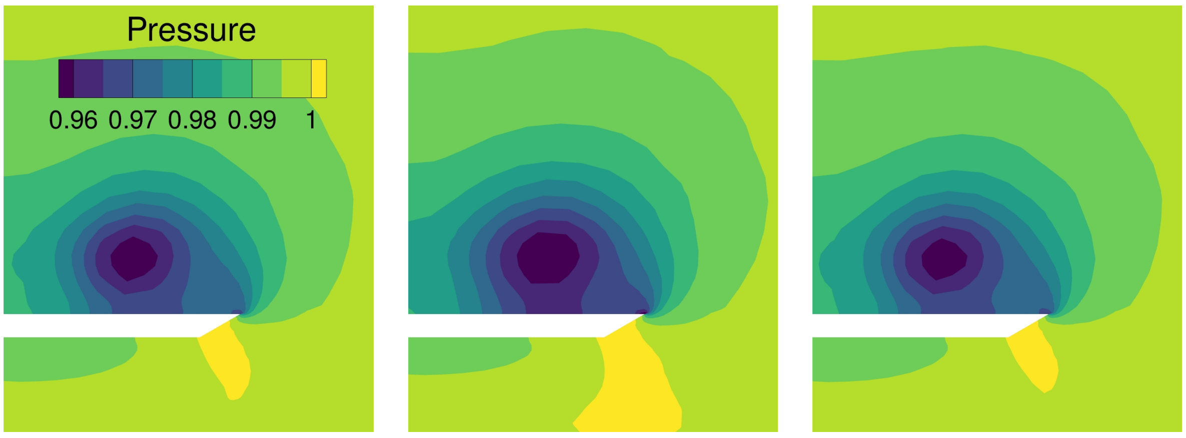
Correction Results

First results depicting the pressure field along the cut at $x = 0.925$ is shown in Figure 5.16 for the $p = 0$ correction at Mach number $M = 0.6$ and angle of attack $\alpha = 15.0°$. As previously shown, the difference between $p = 0$ and $p = 2$ is significant, as former does not lead to vortex shedding. Applying the RF correction, the pressure field highly resembles the high-fidelity solution obtained with $p = 2$. The same holds true for the $p = 1$ correction given in Figure 5.17, presenting the pressure field at the same cut location but for test sample at Mach



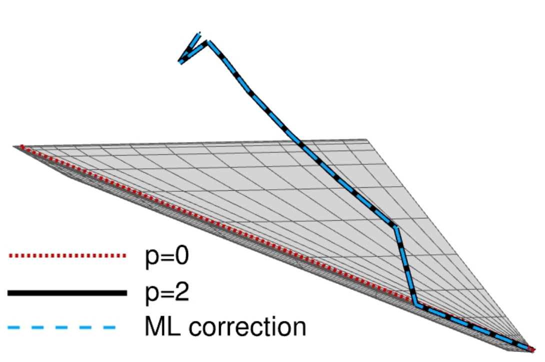
(a) $p = 2$ (b) $p = 0$ (c) RF correction

Figure 5.16: Pressure field along cut at $x = 0.925$ at $M = 0.6$ $\alpha = 15.0°$, delta wing

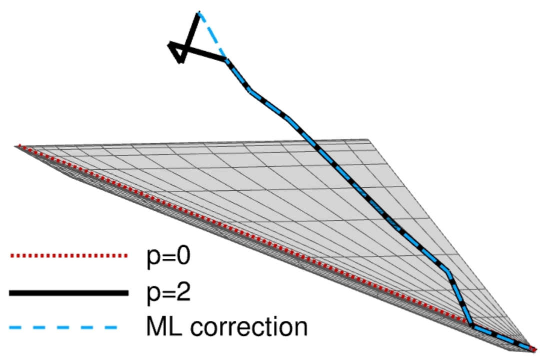


(a) $p = 2$ (b) $p = 1$ (c) RF correction

Figure 5.17: Pressure field along cut at $x = 0.925$ at $M = 0.4$ $\alpha = 13.0°$, delta wing

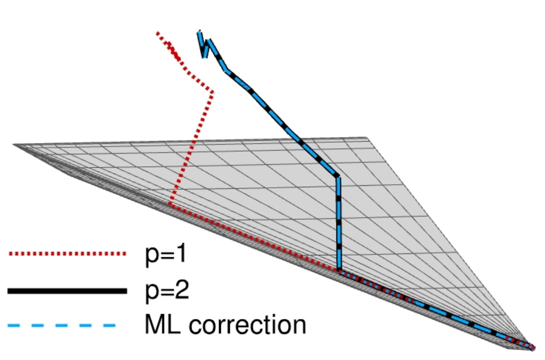


(a) $\alpha = 18.0°$

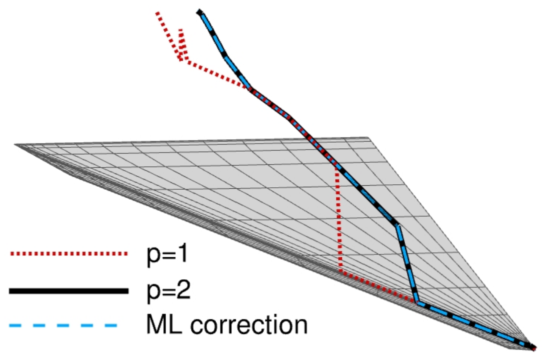


(b) $\alpha = 20.0°$

Figure 5.18: Vortex trajectories based on minimum pressure criterion for $p \in \{0, 2\}$ and ML correction at $M = 0.4$, delta wing



(a) $\alpha = 16.0°$



(b) $\alpha = 21.0°$

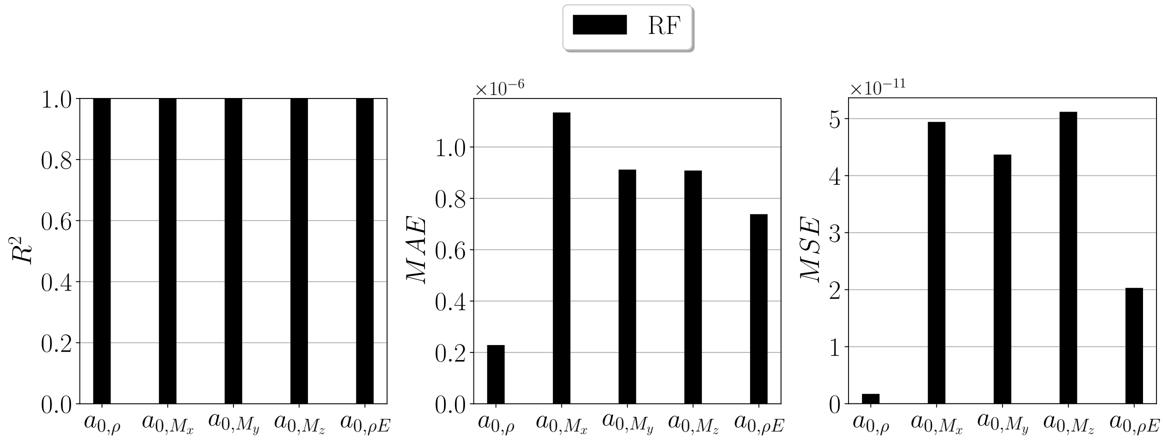
Figure 5.19: Vortex trajectories based on minimum pressure criterion for $p \in \{0, 1\}$ and ML correction at $M = 0.6$, delta wing

number 0.4 and angle of attack 13.0°. The differences between $p = 1$ and $p = 2$ are not as significant, but the RF correction ultimately improves the lower-fidelity solution.

Using a minimum pressure indicator to find the core of a vortex, its trajectory can be visualized, as given in Figure 5.18. For both presented test samples, the $p = 0$ simulation does not result in vortex shedding, which is indicated by the red line following the leading edge. On the other hand, the high-fidelity simulation with $p = 2$ starts to shed a vortex relatively early, more so for the higher angle of attack on the right hand side. On the left, for test sample at Mach number $M = 0.4$ and angle of attack 18.0°, the vortex trajectory based on the RF correction perfectly matches the high-fidelity trajectory. For slightly higher angle of attack 20.0°, deviations are visible towards the trailing edge of the delta wing. Nevertheless, the correction overall improves the low-fidelity vortex trajectory.

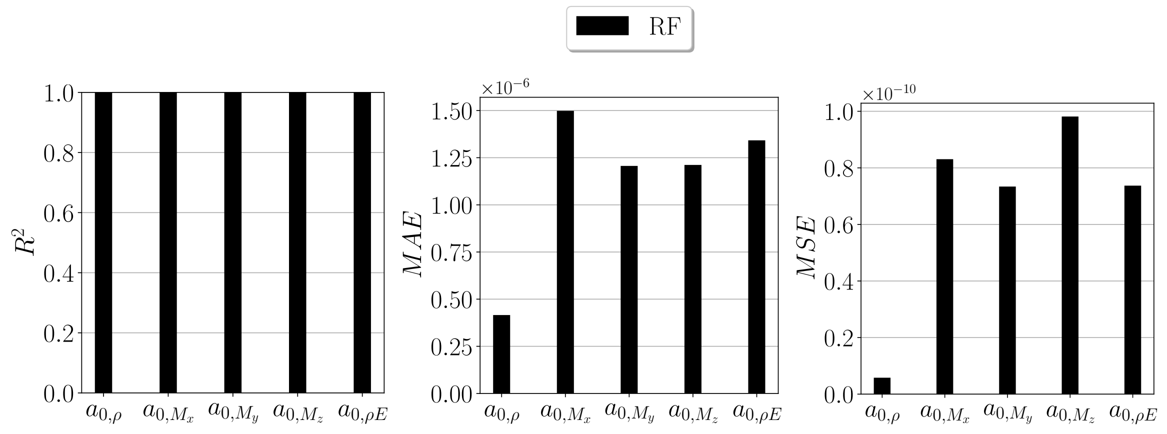
Similarly, Figure 5.19 shows the vortex trajectories for $p = 1$ simulations and the respective corrections. On the left, results for sample point at Mach $M = 0.6$ and angle of attack $\alpha = 16^\circ$ are shown, and on the right results for $M = 0.6$ and angle of attack $\alpha = 21^\circ$ are presented. This time, it is evident for both cases that vortex shedding starts for the lower polynomial degree. Nevertheless, the point of detachment starts later for $p = 1$ compared to $p = 2$. This behavior is corrected by the RF predictions, resulting in vortex trajectories matching the $p = 2$ trajectories.

Finally, Figures 5.20 and 5.21 depict the resulting metrics on the test sets, computed between the predicted correction and true correction, including coefficient of determination R^2, MAE, and MSE. First of all, it is visible how significantly high the R^2 and low the MAE and MSE metrics are. For the test set at Mach number $M = 0.4$, the MAE ranges between 2.26×10^{-07} and 1.13×10^{-06}, while



(a) R^2 (b) MAE (c) MSE

Figure 5.20: Test set metrics for $p = 0$ and $M = 0.4$, delta wing



(a) R^2 (b) MAE (c) MSE

Figure 5.21: Test set metrics for $p = 0$ and $M = 0.6$, delta wing

Table 5.8: Worst metrics R^2, MAE, and MSE on test sets for $p = 1$, delta wing

	$\boldsymbol{R^2}$		**MAE**		**MSE**	
$M = 0.4$	0.994	a_{3,M_x}	3.61×10^{-7}	a_{0,M_z}	2.06×10^{-11}	a_{0,M_z}
$M = 0.6$	0.996	a_{3,M_x}	4.68×10^{-7}	a_{0,M_x}	2.98×10^{-11}	a_{3,M_y}

the MSE values are even lower. The minimum R^2 value here is found at 0.9987 for the polynomial coefficient for the density. Slightly higher values for MAE and MSE are encountered for the test set at higher Mach number $M = 0.6$, although they are overall still extremely low compared to the previous 2D case. Although this test case, the delta wing, required significantly more simulation samples, it can be concluded that learning the correction for the here encountered flow conditions is an easier task than for the previously encountered cases. This, as only laminar flow is considered and no significant discontinuities appear within the design space. This explains the comparable high R^2, as well as the low MAE and MSE values. Additionally, this aligns with the fact that the RF model performed better than the GNN model, as latter showed especially improved results for conditions exhibiting discontinuities.

Similar results are obtained across all polynomial coefficients for the $p = 1$ correction. Since featuring a figure with all metrics does not add any significant value to this evaluation, table 5.8 reports only the worst values for both test sets.

5.3 Limitations

The previously presented results highlight the correction capabilities of the trained models. Nevertheless, as with the FV corrections shown in the previous chapter, the proposed correction approach for the DG discretization exhibits several limitations, which are shown in the following. Limitations posed by the low-fidelity discretization itself are discussed, followed by restrictions given by the data-driven modeling approach.

5.3.1 Low-Fidelity Discretization: Truncation Error

Since the correction term is defined with respect to the truncated high-fidelity solutions, as given in (3.9), the ML corrected simulations can ultimately only approximate the truncation given in (3.8). This bears the question of how large the error between high-order solution $\tilde{\boldsymbol{u}}_{HO}$ and its truncation $\tilde{\boldsymbol{u}}_{T,LO}$ is. As the truncation error ε_T arises due to the omission of the high-order polynomial coefficients, it can be quantified as:

$$\varepsilon_T = \tilde{\boldsymbol{u}}_{HO} - \tilde{\boldsymbol{u}}_{T,LO} = \sum_{i=N_{LO}+1}^{N_{HO}} a_{i,HO} \boldsymbol{\phi}_i. \tag{5.1}$$

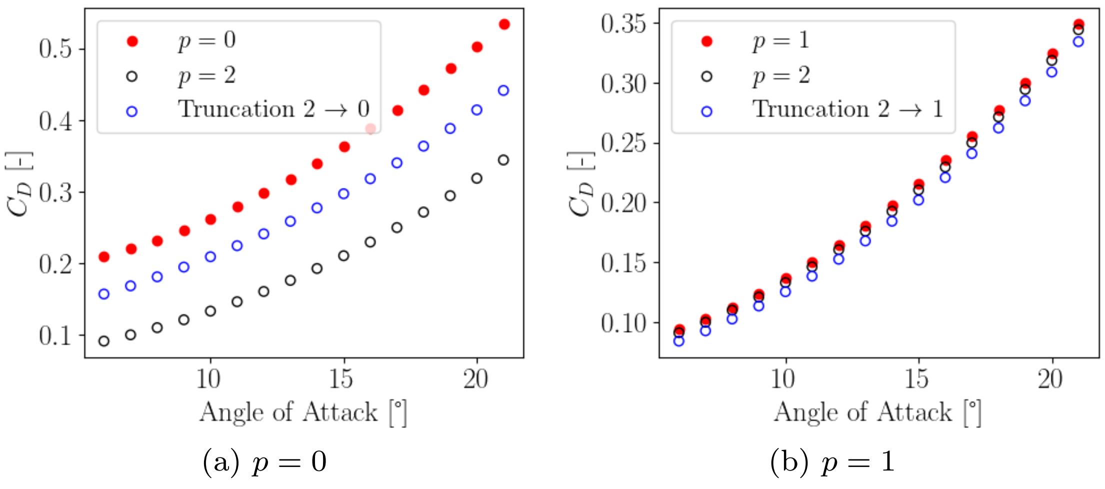


Figure 5.22: Drag coefficient C_D for test set at $M = 0.4$, delta wing

For previously presented results, including the lift and pressure coefficient for the RAE2822 airfoil, as well as the pressure field and vortex trajectory for the delta wing, this error showed no significant influence. Other values of interest are more effected by this error, here we exemplarily show the drag coefficient C_D for the delta wing in Figure 5.22. Values requiring evaluations of derivatives, such as C_D, are strongly influenced by the truncation error, as derivatives magnify the error. First of all, Figure 5.22 shows a gap between the solutions obtained with $p = 0$ and $p = 2$. Truncation leads to an improvement of the drag coefficient, but a gap remains, due to ε_T. For $p = 1$, the truncated solution similarly fails to approximate the drag coefficient obtained on the $p = 2$ discretization.

Furthermore, Figure 5.23 depicts the lift coefficient C_L for the test set at Mach number $M = 0.6$ for the delta wing case. It presents the C_L values for low-fidelity

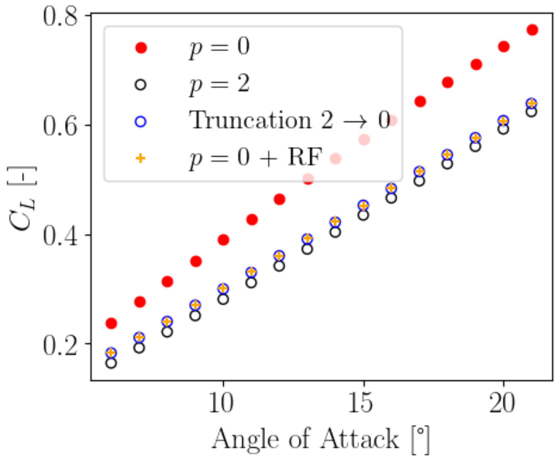


Figure 5.23: Lift coefficient C_L for test set at $M = 0.6$ for $p = 0$, $p = 2$, truncation of $p = 2$ to $p = 0$, and RF correction

simulation obtained with $p = 0$, high-fidelity simulation $p = 2$, the truncation of low-fidelity solution onto the lower polynomial degree discretization, and the RF correction. First of all it is again visible that a gap arises between truncation and high-fidelity simulation, which was previously not the case for lift coefficient values of the 2D RAE2822 DG test case. This additional gap can be explained due to the relatively stretched elements on the wing surface, not sufficiently preserving the high-fidelity accuracy. The RF correction matches well the truncation. Thus, the main limitation of accuracy is here posed by the low-fidelity discretization itself.

5.3.2 Data-Driven Modeling Restrictions

For the FV chapter, the limitations posed by the data-driven method was presented first in terms of generalization capabilities. For the DG study, the same generalization limits hold, and are thus not repeated. Instead, the physical soundness of the corrected solutions will be discussed. After this, a short overview of the time cost will be presented, summarizing the time needed to generate data and train the models.

Preservation of Continuity

As discussed in the simulation setup of the delta wing, additional samples were introduced at high and low angles of attack to address a decrease in accuracy observed in the validation samples. This decline of accuracy was not reflected in the validation metrics, including R^2, MAE, or MSE. Instead, nonphysical values were obtained in the corrected solution, namely negative density in several elements. By adding the 72 additional samples via full factorial sampling, as described in 5.2.1, these occurrences were avoided. This outcome highlights a fundamental limitation of a purely data-driven approaches, which do not prevent nonphysical predictions. Potential strategies to mitigate this issue include incorporating positivity preservation techniques or enforcing physical constraints during model training.

Data Preparation and Training Time

Table 5.9 summarizes the number of simulation samples for the RAE2822 and the delta wing, as well as the resulting simulation time. The time is computed by using the reported average wall clock times given in table 5.1 and 5.5 for the simulations of the high and the respective low polynomial degrees, as well as the time needed to train the final ML models. For the RAE2822, the training time for the GNN is chosen. Needing on average 50 minutes for a $p = 2$ simulation of the RAE2822, the break even point, after which the proposed approach would be beneficial, would be reached after 62 simulations using the $p = 0$ correction and after 68 simulations using the $p = 1$ correction. For the delta wing, requiring 7 hours and 40 minutes for one high-fidelity simulation, the break even point would be reached after 174 and 185 simulations for the $p = 0$ and $p = 1$ corrections, respectively.

Table 5.9: Number of training samples and total time needed for data collection and final ML model training, DG study

	RAE2822		**Delta wing**	
	$p = 0$	$p = 1$	$p = 0$	$p = 1$
No. of samples	60	60	172	172
Simulation time	50.5 h	55 h	1'321 h 32 min	1'367 h 24 min
Training time	45 min	1 h	9 h 25 min	47 h
Total time	51 h 15 min	56 h	1'330 h 57 min	1'414 h 24 min

Especially from the delta wing test case it is obvious that the main factor of computational cost stems from conducting high-fidelity simulations. Nevertheless, as previously emphasized in the FV chapter in section 4.3.2, it can be expected that in industrial settings, in which a surrogate model for quick evaluations might be beneficial, simulation data might already be available. The significant increase in training time of 47 hours for the RF for the $p = 1$ correction of the delta wing could further be improved. This, by restricting the depth of growth for each decision tree or reducing the number of features considered for a split, although these measures might simultaneously reduce the model's performance. Another point not leading to performance degradation would be to increase the number of parallel workers to fit the decision trees.

5.4 Concluding Remarks

This chapter presented a procedure to quantify the discretization error between DG solutions obtained on the same grid but with different polynomial degrees. Subsequently to training ML models on collected data, they are employed to infer a correction term. As for the FV chapter, the RF and GNN models show on the RAE2822 improved results for flow field variables, pressure coefficient, and lift coefficient. Differently to the previous FV chapter, the GNN does not present superior performance with regard to the pressure coefficient, although the overall metrics, such as the coefficient of determination, suggest that the GNN prediction are of higher accuracy than the RF ones. Additionally, where shocks are present, the GNN corrections are better than the RF ones. The procedure was also applied to laminar flow around a delta wing, with the aim of assessing how well the corrections reconstruct the vortex shedding phenomenon. As a greater design space was considered, and nonphysical corrections were encountered at high and low angles of attack, more sample points for training are needed. The corrected flow fields and vortex trajectories show significant improvements compared to the low-fidelity baseline simulations.

On the one hand, limitations of the proposed method were encountered for certain values, such as the drag coefficient. Additionally, a major limitation of the proposed method is that the corrections do not necessarily preserve the physical soundness of the simulation results, leading to negative values of density. On the other hand, the significant improvement of accuracy, as was for example shown for the vortex shedding, highlight that the discretization error with respect to a low polynomial degree DG simulation can be quantified, learned and finally corrected. Learning such corrections for a DG discretization allows to use a single grid, simplifying the data generation. In the following, this will be exploited for the correction of unsteady DG simulations.

Chapter 6

Application: Correction of Unsteady DG Simulations

To accurately resolve unsteady problems, a sufficiently resolved spatial and temporal discretization is needed to account for relevant timescales, which renders these simulations a computationally expensive task. Compared to high-order DG solutions, the low-order ones require fewer degrees of freedom in space, evaluations at fewer quadrature points, and allow for larger time steps, thereby considerably reducing the computational effort. However, the trade-off is in general lower accuracy.

Consider for example the flow around a 2D cylinder simulated with varying polynomial degree p employing the same hardware, with the density field given in Figure 6.1. $p = 0$ results in a steady state, not being capable to develop the von Kármán vortex street behind the cylinder. The solution with $p = 1$ displays vortex shedding with a time step size of $\Delta t = 0.5$, whereas $p = 2$ requires a smaller time step $\Delta t = 0.1$ for robustness. Simulating 500 seconds, starting from freestream conditions, requires around 11 minutes and 3.5 hours for $p = 1$ and $p = 2$, respectively. Nevertheless, only the most expensive one is accurate, judged by the vortex shedding interval of $t_s = 43s$, deviating only by milliseconds from the analytical value of $t_s = 42.5s$. While the cylinder is only a simple showcase, it nevertheless suggests that correcting unsteady low-fidelity simulations could

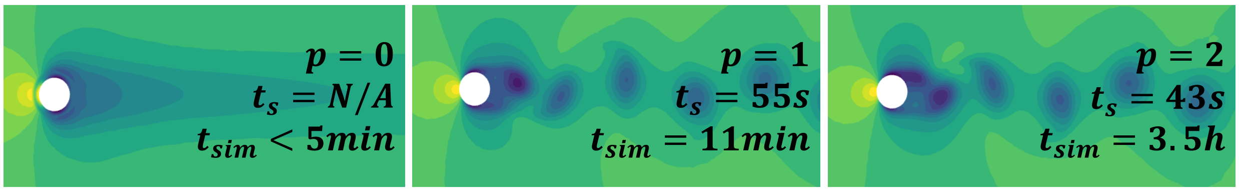


Figure 6.1: Flow around a 2D cylinder across varying polynomial degrees $p \in \{0, 1, 2\}$

offer a promising approach to make the simulation of unsteady phenomena more accessible.

This chapter focuses on the correction of low-order DG simulations for smooth transport problems, avoiding discontinuities to concentrate on the added complexity of time dependency. The DG discretization is a reasonable choice for smooth problems and allows to obtain solutions with varying accuracy on the same grid. Two different correction approaches and two different ML training methods are investigated, as previously described in section 3.3. The first correction method is a post-processing approach, in which the ML correction is applied completely decoupled from the CFD solver iterations. Employing a GNN model and similar input features, this approach builds mainly on the steady state work presented in the previous chapter 5. The GNN is chosen since the NN has previously not performed as well as the RF or the GNN, and the RF is not employed since the RL approach requires a deep learning model.

The main goal is to extend the correction approach from steady to unsteady simulations. The second correction approach investigates the robustness and accuracy of an ML correction which is applied after each solver iteration, comparable to [42, 118]. Similar investigations are conducted for the ML training: first, a decoupled ML training approach is conducted, by simply using supervised training in an offline fashion. Secondly, an online training method is tested by employing the PPO RL algorithm which interacts with the CFD solver during training and prediction. With this, the aim is to investigate if RL is a viable tool to couple ML with a CFD solver, and thus to answer the question if RL is a potential alternative to differentiable solvers for the application of unsteady flow field corrections.

Overall, the generalization capabilities of the models are assessed in two ways: firstly, in terms of different parameters within the design space, and secondly, in terms of providing long-term stable corrections at time instances beyond the training scope. The content of this chapter has previously been presented in [84].

6.1 1D Case - Linear Advection

The aim of using a simple 1D case is to test and compare the methodologies presented in section 3.3 and to assess their generalization capabilities. For this, the linear advection of a sine wave is investigated, which is described for a quantity u and non-zero constant velocity coefficient c by the PDE given in (2.11).

6.1.1 Data Generation

In the following, the simulation setup is presented, including a description of the design space and the sampling strategy. Subsequently, simulation results obtained with varying polynomial degrees are shown, displaying the dissipative nature of low-fidelity simulations, which is intended to be corrected by the ML predictions.

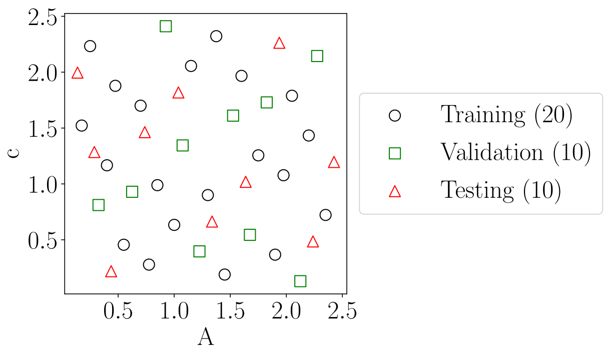


Figure 6.2: Design of experiment for 1D linear advection samples

Simulation Setup

To approximate the solution of the boundary value problem, a modal DG discretization is chosen with a third order RK time discretization. The domain from $x = 0$ to $x = 1$ is discretized with 100 equally spaced elements. For the implementation, a code is written in Python, employing Legendre polynomials, Gauss quadrature points and upwind flux for linear advection. A sine wave is implemented for the initial condition and the domain uses periodic boundary in flow direction. To generate parametrized data, simulations are carried out across varying velocity coefficients c and amplitudes A. For this, a design of experiment is created with a Halton sequence, as given in Figure 6.2, using the SMARTy library [13]. The 20 black samples are used for the training of all models, and the 10 green samples for the validation of the models trained with supervised learning. For training and validation, the time span $0 \leq t \leq 0.5$ is considered, resulting in 100 snapshots per sample, since a time step of $\Delta t = 0.005$ for the low-order trajectories is kept constant. For testing, not only the red samples in Figure 6.2 are considered for an extended time $0 \leq t \leq 5.0$, but also the train and validation samples within the time span $0.5 < t \leq 5.0$, with the goal of evaluating the behavior of the models during temporal extrapolation.

Simulation Results

Simulations are conducted with polynomial degree $p \in \{0, 1, 2\}$ for a sine wave with amplitude $A = 2.0$ and velocity coefficient $c = 1.5$. Results obtained at $t = 0.5$ and $t = 5.0$ seconds are presented in Figure 6.3. It is visible that the solution with $p = 0$ is inaccurate, resulting in a discretization error which induces high numerical diffusion, leading to an increase of the discretization error and thus a decrease of the amplitude with time. Increasing the polynomial degree leads to solutions which are sufficiently accurate and maintain the amplitude. With this, the goal of this test case is to investigate if solutions with $p = 0$ can be corrected with the aforementioned methods described in section 3.3.

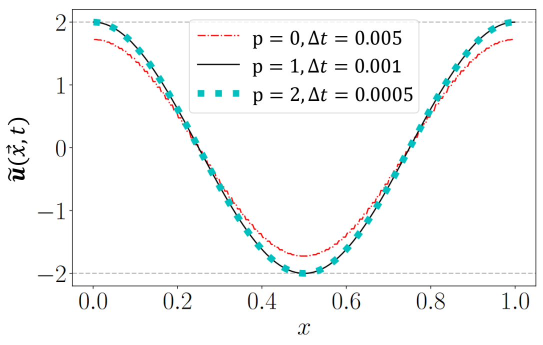
6.1.2 Machine Learning Correction

In this section, the ML training approaches are descibed. Then, the ML corrected solutions are shown, first qualitatively on a selected test case, and then quantitatively across the whole design space at different time steps.

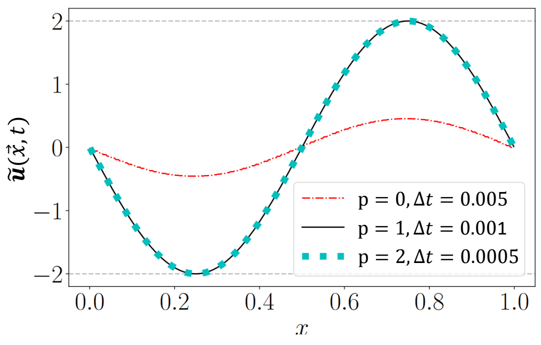
In general, all ML methods follow an element-wise correction, that is the model predicts for each element separately a correction. The underlying ML model for all correction methods is a GNN with convolutional layers to incorporate information from the surrounding neighborhood elements. The graph representation is depicted in Figure 6.4. Each element k of the grid corresponds to a graph vertex v_k inside the set and is connected to its neighboring vertices $v_j \in N(v_k)$ by the edges $e_{k,j}$. Note that first and last vertices are also connected, representing the periodic boundary condition. The feature inputs shared across all methods are the amplitude A, velocity coefficient c, and the first and second derivative of the degrees of freedom $\boldsymbol{a}$ of the current element at the current time step t and at the previous time step $t - \Delta t_{LO}$. Thus, the feature vector for element k is

$$\eta(\tilde{\boldsymbol{u}}_{\boldsymbol{LO}}) = \left[A, c, \frac{\partial \boldsymbol{a}_{LO}}{\partial x_i}\bigg|_{k,t-\Delta t}, \frac{\partial^2 \boldsymbol{a}_{LO}}{\partial x_i \partial x_j}\bigg|_{k,t-\Delta t}, \frac{\partial \boldsymbol{a}_{LO}}{\partial x_i}\bigg|_{k,t}, \frac{\partial^2 \boldsymbol{a}_{LO}}{\partial x_i \partial x_j}\bigg|_{k,t}\right]. \quad (6.1)$$

The use of information from the previous time step allows to embed temporal information. It was found that adding additional prior time steps did not increase the accuracy of the prediction, but only leads to an increased feature dimension



(a) $t = 0.5s$



(b) $t = 5.0s$

Figure 6.3: Solution $\tilde{u}$ for wave with amplitude $A = 2.0$ and velocity $c = 1.5$ for polynomial degrees $p \in \{0, 1, 2\}$

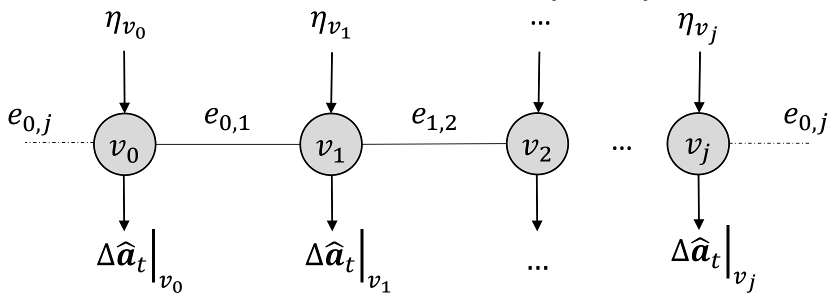

η_{v_0}
η_{v_1}
...
η_{v_j}
$e_{0,j}$
$e_{0,1}$
$e_{1,2}$
$e_{0,j}$
v_0
v_1
v_2
...
v_j
$\Delta\hat{\boldsymbol{a}}_t\big|_{v_0}$
$\Delta\hat{\boldsymbol{a}}_t\big|_{v_1}$
...
$\Delta\hat{\boldsymbol{a}}_t\big|_{v_j}$

Figure 6.4: Graph representation

and higher computational cost, especially during training. The output of the network trained with supervised learning is the element-wise correction $\Delta\hat{\boldsymbol{a}}_t$ for the current time step t. For the RL approach, two GNNs are implemented, the policy network and the critic network. In the RL framework, the described features are the state observations, which the policy network receives as input. Based on this, it predicts the mean μ of the distribution of the correction $\Delta\hat{\boldsymbol{a}}_t$, while the value network predicts how good it is to be in a given state given the same inputs.

Supervised Learning Setup

To ensure a fair comparison between the two different supervised learning correction methods, i.e. the models trained for a post-processing correction and an iterative correction, a Bayesian hyperparameter optimization based on minimizing the validation MSE is conducted within the same hyperparameter search space. The results of this search are reported in table 6.1. The supervised learning model requires more layers. A possible contribution to this is the wider spread of the corrective values to be learned. Besides this and the different correction to be predicted, both training methods employ the same hyperparameters. These include an exponential decay of the learning rate and the same loss function, chosen to be the MSE minimizing the difference between the predictions $\Delta\hat{\boldsymbol{a}}$ and the ground truth corrections $\Delta\boldsymbol{a}$ on the training data.

Table 6.1: Hyperparameters for supervised learning models, 1D test case

Hyperparameter	**Post-processing**	**Iterative**
No. of layers	5	2
No. of hidden channels	19	12
Initial learning rate	0.0013	0.0082

Furthermore, the same model with hyperparameters as described above, is trained twice for the iterative correction approach. Once without and once with Gaussian noise added to features related to the degrees of freedom, namely the derivatives. The noise is applied for each batch separately and additionally standardized. The architecture and training settings are kept the same for both models, as given in table 6.1. To find an appropriate amount of noise another hyperparameter optimization for the standard deviation of the Gaussian noise is conducted to minimize the validation MSE. The final validation MSE values for all models trained under supervised learning are shown in table 6.2.

Table 6.2: Validation metrics for supervised learning approaches, 1D test case

Correction approach	**MSE**
Post-processing	3.07×10^{-3}
Iterative	1.30×10^{-3}
Iterative with noise	0.98×10^{-3}

Reinforcement Learning Setup

To drive the RL agent, the reward function is chosen as the squared error between the truncated high-order solution and the corrected solution, such that:

$$r_k(t) = -(\tilde{u}_{T,k}(t) - \hat{u}_k(t))^2. \tag{6.2}$$

This value is computed for each element k after each time step Δt, thus being a non-sparse reward. Note, the leading negation of the function ensures that the difference is minimized, since the reward is maximized by definition. Correcting a solution with polynomial degree $p = 0$ with basis function $\phi_1 = 1$, (6.2) simplifies to

$$r_k(t) = -(a_{1,k,HO}(t) - (a_{1,k,LO}(t) + \Delta\hat{a}_{1,k}(t)))^2. \tag{6.3}$$

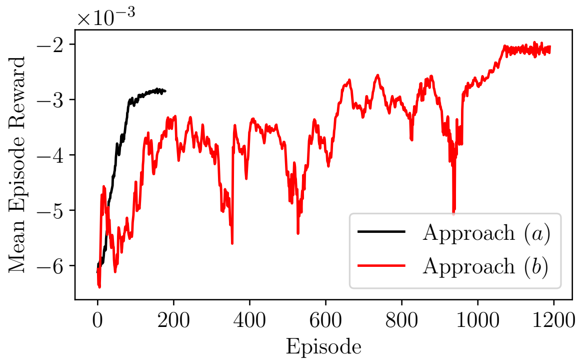
With this, the reward is comparable to the loss function of the models trained using supervised learning, allowing to evaluate the final results under similar training conditions. Two approaches, denoted as (a) and (b), are investigated to find optimal hyperparameters.

For approach (a), no initial guess is taken for the network architectures, and most of the hyperparameters are tuned using Bayesian optimization. The number of maximum episodes is restricted, such that the training stops when reaching 200 episodes. The training also concludes before if the reward converges. The tuned hyperparameters include the standard deviation for the action distribution, the number of hidden channels for the actor and critic network, the number of convolutional layers, the activation function, learning rate, the discount rate, the number of time steps per batch, the number of network updates per episode, the clipping factor ϵ, and the entropy coefficient.

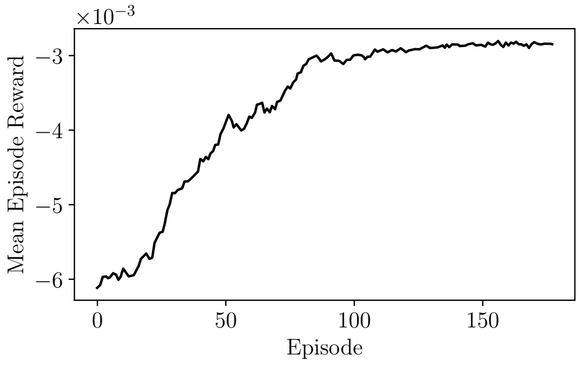
For approach (b), the same network architecture is used for the actor network as the one found for the iterative correction model with supervised learning. Regarding hyperparameters, several best practices from [5] are followed. These include initializing the weights of the last policy network layer to a small value, no use of entropy for exploration, and having a wider critic network compared to the actor network. For this, the number of hidden channels of the critic network is set to 128. Then, only the standard deviation for the action distribution, the learning rate, the number of time steps per batch, and the number of network updates per episode are tuned. The number of maximum episodes is not restricted, such that the training concludes as soon as the reward converges.

No validation set is used during training, typical for RL, as the observed states behave non-stationary. Thus, the agents only encounter the 20 simulations from the training set, which are all re-computed for each episode. The actor prediction is used as correction after each solver iteration and the reward can be computed according to (6.3). Figure 6.5 plots the resulting mean rewards for both training approaches. First, employing a maximum number of episodes for (a) results in early convergence of the reward, while approach (b) requires a total of 1'889 episodes to conclude.

The experienced mean rewards for approach (a) and (b) are given in Figure 6.5a. The reward for approach (a) with extensive hyperparameter optimiza-



(a) Reward of (a) and (b)



(b) Reward of (a)

Figure 6.5: Mean reward across episodes for agent with (a) extensive hyperparameter search with limited number of episodes, and (b) initial guess for the model architecture

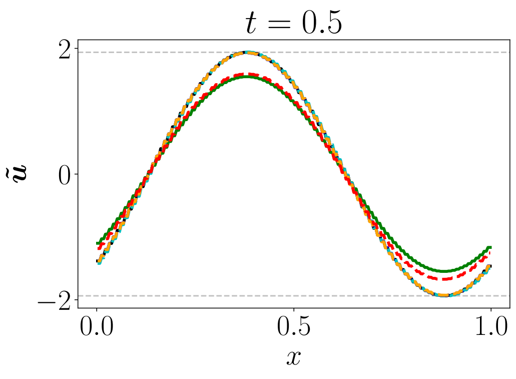
tion and limitation of maximum number of episodes again presented in Figure 6.5b for better visualization. The initialization of the last layer does not seem to have much effect, as both agents start in the first episode with a similar reward around -6×10^{-3}. The agent which was tuned more extensively and was limited in the number of maximum episodes converges faster to a steady mean reward. Here, the training stops at episode 177. The approach using initial guesses leads to more exploration, visible by the noisy mean reward across the episodes in Figure 6.5a. This leads to the agent needing more episodes, a total of 1'889 to be precise, until convergence. Nevertheless, this results in finding a slightly higher final mean reward. The final magnitude of the mean reward is comparable to the final validation MSE of the models trained under the supervised learning framework - although it has to be emphasized that for the RL agents, this value is based on the training data as given in (6.3).

The agent exhibiting noisier behavior uses no entropy factor and the hyperparameter tuning resulted in a lower standard deviation for the action distribution, favoring less exploration. The smoother progress of the other model can be explained by the limited number of episodes, restricting the agent to find an optimal policy in a shorter time. Additionally, the model resulting in Figure 6.5b favors a network with more parameters, here with 5 convolutional layers and 13 hidden features, compared to the other one having 2 layers and 12 hidden features. More convolutional layers include more neighborhood information, possibly leading to increased regularization and thus less exploration. Although the reward curves suggest two fairly different agents, similar trends with no significant differences have been found while comparing the results. Thus, only corrections from agent (b) resulting in a higher mean reward are presented in the following.

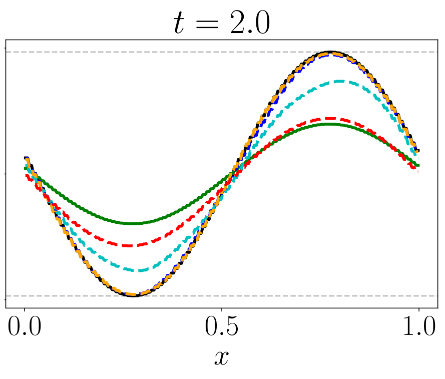
Correction Results

For the ML correction results, first, qualitative findings are presented for the test sample with the highest amplitude $A = 1.9375$ and velocity coefficient $c = 2.263$, given in Figure 6.6. Time steps up to $t = 0.5$, still within the training time window, can be regarded as extrapolation test in terms of amplitude and velocity coefficients, since the sample is not surrounded by training samples in the design space. Any snapshot at times beyond $t = 0.5$ is a test of temporal extrapolation.

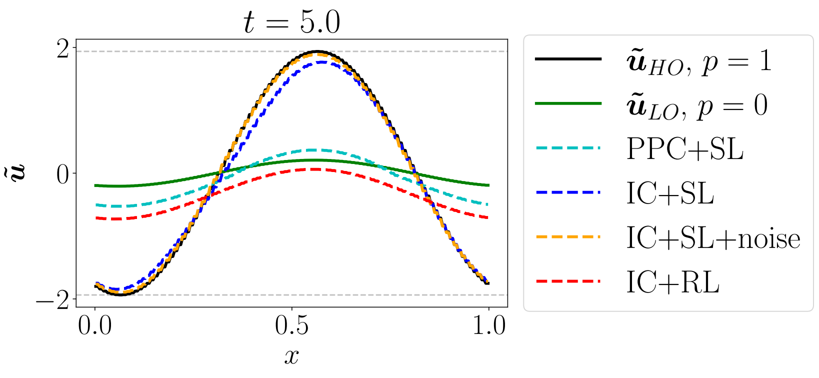
At time $t = 0.5$, equaling 100 passed time steps, all models except for the one trained under the RL framework show excellent results, matching the high-order solution $\tilde{\boldsymbol{u}}_{HO}$ visually very well, as given in Figure 6.6a. The solution corrected by the RL agent is closest to the low-order solution $\tilde{\boldsymbol{u}}_{LO}$, showing only slight corrective capabilities. At $t = 2.0$, after 300 more time steps, the post-processing correction (PPC) degrades. This is expected, since under the post-processing approach the error grows over time as the low- and high-order trajectories diverge, and the greatest discrepancies occur at higher amplitudes and velocity coefficients. The model is not able to extrapolate to such extents, resulting in less effective corrections. The iterative correction (IC) models trained under the supervised learning (SL) framework start to show slight deviations, but still correct the low-



(a) $t = 0.5$



(b) $t = 2.0$



(c) $t = 5.0$

Figure 6.6: Qualitative results for test sample with amplitude $A = 1.9375$ and velocity coefficient $c = 2.263$

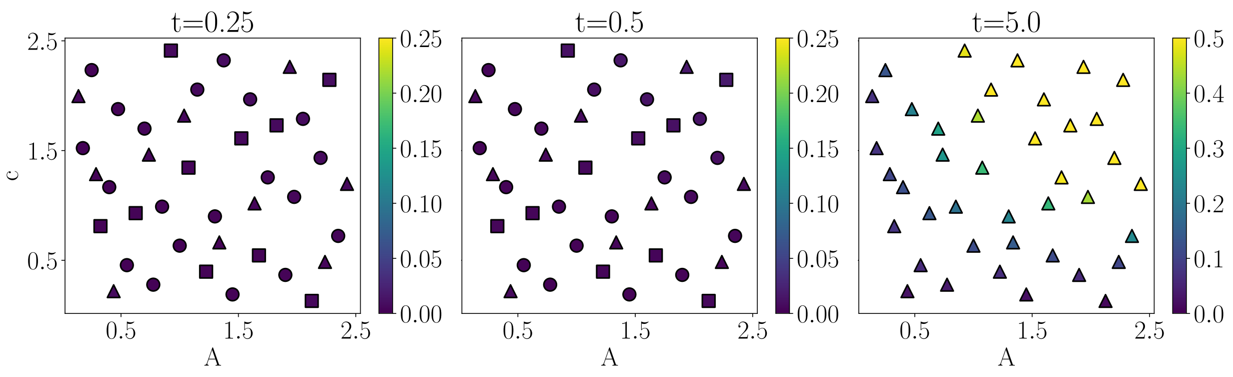


Figure 6.7: MAE across all samples for post-processing correction with supervised learning. Legend: ○ training samples, □ validation samples, △ test samples.

order solution overall. At $t = 5.0$, another 600 steps ahead in time, only the two models trained under supervised learning using an iterative correction achieve to approximate the high-order solution $\tilde{\boldsymbol{u}}_{HO}$. Nevertheless, within these time steps an error accumulation can be observed, and the amplitude of $A = 1.9375$ is shrinking, distorting the initial wave. The model trained with regularizing noise shows improved corrections compared to the one trained without noise.

A quantitative evaluation of the corrective capabilities within the design space is given in Figures 6.7, 6.9, and 6.8. Here, the MAE is plotted for each model at three time steps and is defined as

$$\mathrm{MAE} = \frac{1}{K}\sum_{i=1}^{K} \left|\tilde{u}_{i,T} - \hat{u}_i\right|, \tag{6.4}$$

with $K = 100$ denoting the number of elements in the grid. For each snapshot at the indicated time, the MAE is computed across all elements, displaying the difference between the ML corrected solution $\hat{\boldsymbol{u}}$ and the truncated solution $\tilde{\boldsymbol{u}}_T$. Note the different scale for the last time step and that there are no validation samples for the RL model, as denoted by the legend. Furthermore, since training

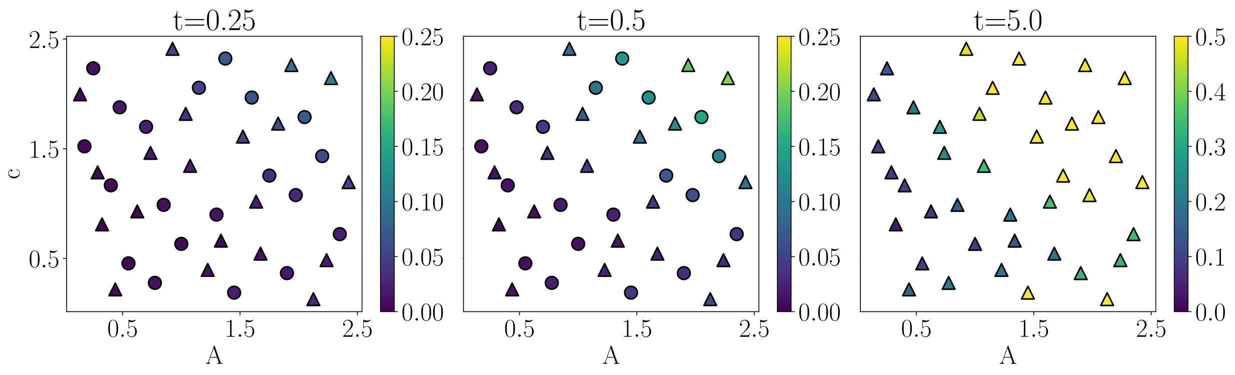
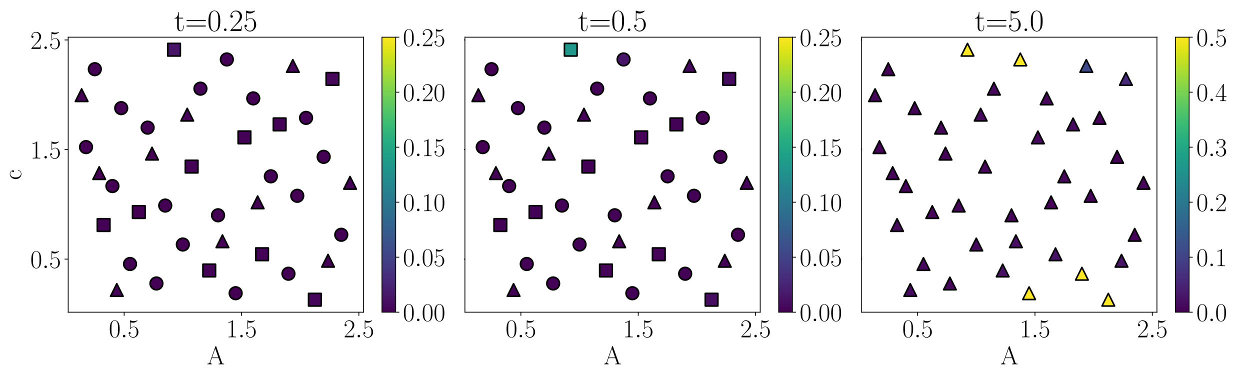


Figure 6.8: Iterative correction with RL: MAE across all samples. Note that there is no validation data. Legend: ○ training samples, △ test samples.

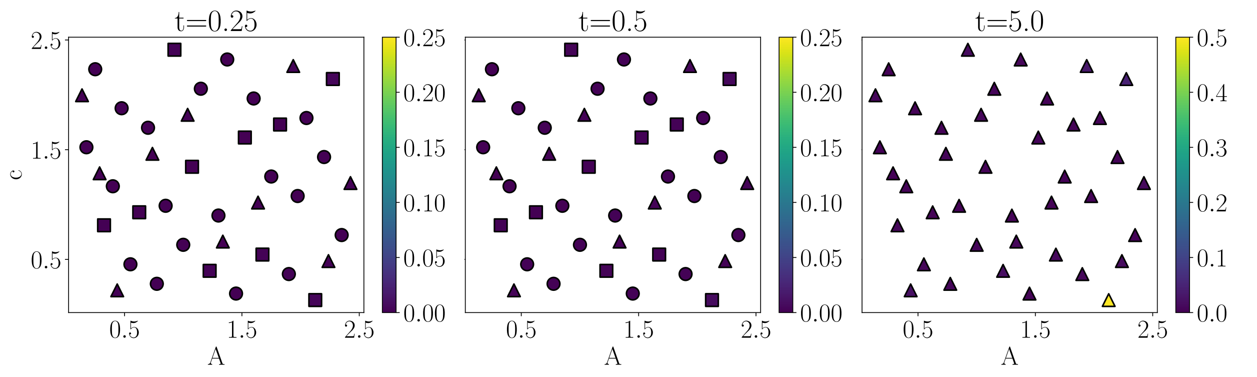
and validation data is only considered up to $t \leq 0.5$, all samples are labeled as test sample at the final time $t = 5.0$.

For the post-processing correction trained with supervised learning, Figure 6.7 shows a low MAE across all training, validation and test samples inside the training time window up to $t = 0.5$. As was already shown in Figure 6.6 with the qualitative results, it is also visible in the MAE that at later times, the post-processing model fails to predict a sufficient correction. This is especially the case for higher amplitudes and velocity coefficients, as the differences between low and high-order solutions increase faster.

For the RL agent, a similar trend as for the post-processing correction model is visible, as depicted in Figure 6.8. The agent fails for samples with high amplitude A and velocity coefficient c, but even earlier during training times, already visible after 50 time steps at $t = 0.25$. As the error accumulates for iterative correction approaches, the MAE grows from left to right for these samples located in the upper right quadrant, where high amplitudes A and velocity coefficients c are located. Additionally, the agent starts to exhibit increased MAE at high amplitudes and low velocities at later time steps. Thus, it can be stated that the online training approach using RL does not increase robustness leading to long-term



(a) MAE for iterative correction with supervised learning



(b) MAE for iterative correction with supervised learning and noise regularization

Figure 6.9: MAE across all samples for iterative correction with supervised learning: (a) without noise, (b) with noise regularization. Legend: ○ training samples, □ validation samples, △ test samples.

stability. On the contrary, the generalization capabilities to adapt to new cases and into longer time spans are overall the poorest, including on training samples. This is an interesting result, since it was expected that RL as an online learning approach would incorporate the autoregressive errors during training, ultimately making the agent more robust.

The two models trained under a supervised learning framework to predict an iterative correction show the best long-term corrections, as displayed in Figure 6.9a and Figure 6.9b. For the model trained without noise regularization, the MAE increases significantly for one validation sample with highest velocity coefficient at $t = 0.5$, which is amplified through the iterative correction approach, well visible at $t = 5.0$ in Figure 6.9a. At this time, equaling a 1'000 time steps, the model shows greater errors for samples with high velocity coefficient c as well as for samples with high amplitude A and low c. Adding noise regularization during the training reduces almost all of these errors, as shown in Figure 6.9b, with only one remaining sample at $t = 5.0$ at the border of the design space exhibiting an increased MAE.

6.2 2D Case - Convection of an Isentropic Vortex

Based on the results for the 1D test case, only the two most promising approaches are further investigated: the post-processing correction with supervised learning, showing reliable results within the design space, and the iterative correction with supervised learning trained under noise regularization, providing long-term accurate corrections. To further assess the models capabilities, a more complex test case is investigated. The considered problem is based on [94], originally proposed to test if high-order FV schemes exhibit numerical dissipation or dispersion. By solving the compressible Euler equations, the case considers a uniform flow, which convects a 2D isentropic vortex in one direction. The reader is referred to [94] for detailed information regarding case and initial conditions.

6.2.1 Data Generation

This subsection presents the simulation setup and simulation results obtained with different polynomial degrees.

Simulation Setup

For the grid, a uniform spacing is considered, with 20×20 equally spaced elements along both axes, resulting in a total of 400 grid cells. Periodic boundaries are set in the direction of the flow, from left to right along the x-axis. [94] describes a strong vortex with $\beta = 0.8$, where $\beta = v_A/v_\infty$ is the ratio between the maximum velocity induced by the vortex and the uniform flow velocity. As for the 1D problem, data for training, validation and testing is collected with a design of experiment via a Halton sampling sequence, shown in Figure 6.10. The design space is spanned by vortex strength $\beta = [0.25, 0.8]$ and vortex length $L_v = [0.4, 1.0]$. 15

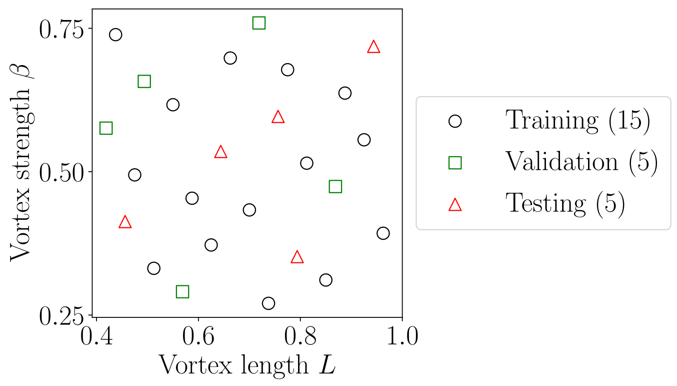


Figure 6.10: Design of experiment for 2D isentropic vortex

simulation samples are used for training, and 5 additional simulations are carried out for the validation and test set, resulting in a total of 25 simulations. The DG simulations are conducted with a Roe upwind scheme for the convective term. A dual time stepping scheme is used to treat the temporal discretization, with linearized implicit Euler employed for the inner iterations, and a diagonally implicit RK scheme for the outer iterations.

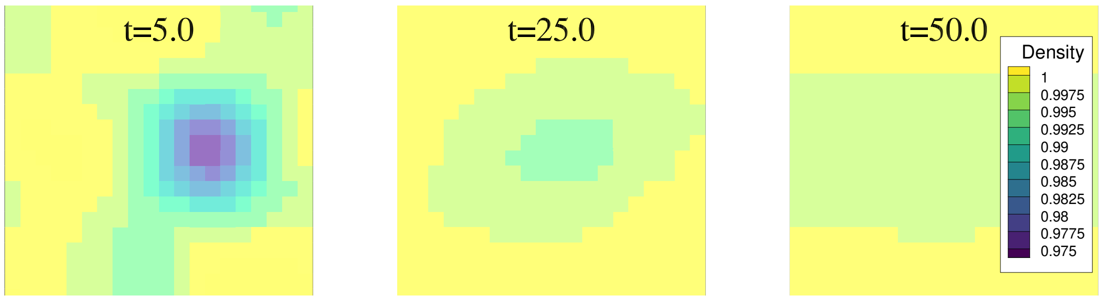
The average wall clock time for simulations up to $t = 25.0$ for different polynomial degrees is given in table 6.3. For all conducted simulations the same hardware is used, that is one node with eight cores and eight threads per core on an HPC cluster. Furthermore, the table reports the total number of degrees of freedom across all elements per equations per time step, highlighting the increased cost for high-order solutions, and the advantage of ML corrections for low-order simulations.

Table 6.3: Average wall clock time for one simulation up to $t = 25.0$, and degrees of freedom per time step and per equation

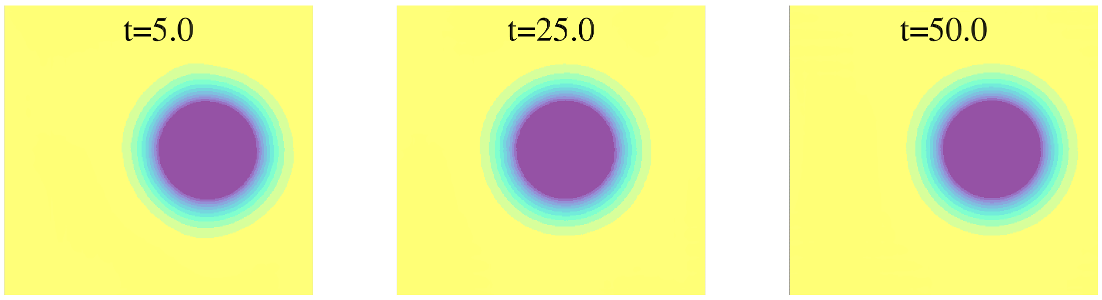
$\boldsymbol{p}$	$\boldsymbol{\Delta t}$	**Time**	**Degrees of freedom**
0	1.0	3 min	100
1	1.0	7 min	300
2	0.5	43 min	800
3	0.5	2 hrs 20 min	1’000

Simulation Results

Simulation results with $M = 0.3$ and vortex length $L_v = 1.0$ are plotted in Figure 6.11 at different times, including $t \in \{5.0, 25.0, 50.0\}$. The density field for polynomial degree $p = 0$ and time step size $\Delta t = 1.0$ is given in Figure 6.11a, and for polynomial degree $p = 2$ with $\Delta t = 0.5$ in Figure 6.11b. Note that for $p = 0$ the solution is a constant across each element, while for higher polynomial degrees, the solution is described as a piecewise polynomial function as given in (2.14). Thus, the solution for $p \geq 1$ can be sampled and displayed at a higher resolution than is given by the computational mesh - which is consistently done



(a) Density fields for $p = 0$ and $\Delta t = 1.0$



(b) Density fields for $p = 2$ and $\Delta t = 0.5$

Figure 6.11: Simulation results for density field at varying times

in this chapter for all vortex visualizations. Figure 6.11 clearly shows that $p = 0$ is inaccurate, leading to significant numerical dissipation within only a few time steps, whereas the solution with polynomial degree $p = 2$ retains the shape of the vortex.

Figure 6.12 plots the density along a cut through the mid-plane of the vortex at $t = 25.0$ for solutions obtained with polynomial degree $p \in \{0, 1, 2, 3\}$. This plot confirms the numerical dissipation induced by $p = 0$, and shows the same effect for $p = 1$, although less significant. Additionally, it can be seen that the center of the vortex is for all solutions located at the same position, thus the vortex retains the desired speed. From this it can be concluded that no dispersion

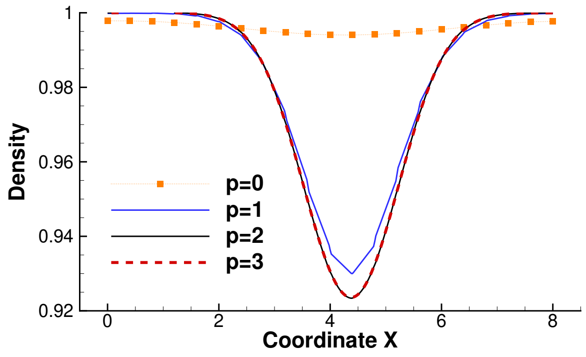


Figure 6.12: Density cut through vortex center for $\beta = 0.8$ and $L_v = 1.0$ at $M = 0.3$ and $t = 25.0$

is encountered, which is to be expected for low wave numbers. Solutions of $p = 2$ and $p = 3$ are equal, indicating convergence with respect to the number of degrees of freedom at $p = 2$. Thus, the goal of this study is to correct the inaccurate solution with polynomial degree $p \in \{0, 1\}$ to approximate the solution obtained by $p = 2$.

This test case adds complexity compared to the previous 1D case by adding another dimension and by correcting both $p = 0$ and $p = 1$. Additionally, with the Euler equations a non-linear problem is solved and the aim is to correct not only one unknown, but rather all variables of interest, such that $\tilde{\boldsymbol{u}} = \{\rho, M_x, M_z, \rho E\}$.

6.2.2 Machine Learning Correction

Since the reinforcement learning approach has not shown any significant advantages on the 1D case, only two of the supervised learning methods are investigated and described in the following. Then, the ML correction results are displayed, first qualitatively by the density plotted along a cut through the vortex on a selected test case, and then qualitatively across the whole design space at different time steps.

Supervised Learning Setup

For each degree of freedom, all models predict one correction, which results in 4 outputs for $p = 0$ and 12 outputs for $p = 1$ per grid element. For the features, the first and second derivative of each degree of freedom is computed, again for each element separately, as well as for current and previous time steps. For the 1D problem, adding the amplitude A and the velocity coefficient c increased the accuracy across all models. For this case, adding vortex strength β and length L_v as features resulted in no significant benefit for metrics of the validation data, and are thus removed from the feature inputs. Therefore, the feature vector for element k is

$$\eta(\tilde{\boldsymbol{u}}_{\boldsymbol{LO}}) = \left[\frac{\partial \boldsymbol{a}_{LO}}{\partial x_i}\bigg|_{k,t-\Delta t}, \frac{\partial^2 \boldsymbol{a}_{LO}}{\partial x_i \partial x_j}\bigg|_{k,t-\Delta t}, \frac{\partial \boldsymbol{a}_{LO}}{\partial x_i}\bigg|_{k,t}, \frac{\partial^2 \boldsymbol{a}_{LO}}{\partial x_i \partial x_j}\bigg|_{k,t} \right]. \tag{6.5}$$

This results in a total of 48 input features for the $p = 0$ correction and 144 for $p = 1$. No extensive hyperparameter tuning was performed for the training of the post-processing correction model. Instead, simple manual tuning of the learning rate, the number of layers, and the number of hidden features quickly reached a sufficiently low validation MSE, and further changes to the hyperparameters did not significantly effect this metric. For the iterative correction model, the same architecture is chosen as used for the post-processing model. Only noise is added based on a grid search, resulting in a reduction of the validation MSE. As for the 1D problem, this noise is added at each epoch for every batch, and for each feature the noise is scaled by subtracting the feature batch mean before a division by the standard deviation of the batch. The standard deviation for noise regularization amounts to $\sigma = 0.0005$ and $\sigma = 0.00025$ for the $p = 0$ and $p = 1$ correction, respectively. The final validation MSE values are reported in table 6.4. On a GPU

NVIDIA A100 unit, the training time for the models revolves around 10 minutes for both $p = 0$ and $p = 1$ corrections.

Table 6.4: Validation metrics for supervised learning approaches, 2D test case

Correction approach	**MSE**
Post-processing $p = 0$	0.052
Iterative with noise $p = 0$	0.030
Post-processing $p = 1$	0.063
Iterative with noise $p = 1$	0.035

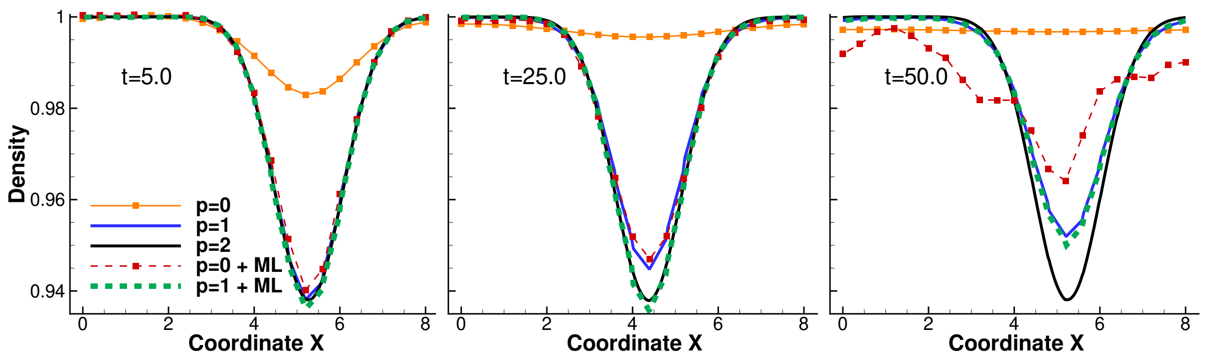
Correction Results

First results are presented on a qualitative basis by plotting the resulting density along a cut through the vortex, given in Figure 6.13 for the test sample with highest vortex strength $\beta = 0.7185$ and length $L_v = 0.9438$. Note that not the corrected degrees of freedom are plotted, but the resulting density. From left to right, results are shown for time steps $t = 5.0$, $t = 25.0$, and finally $t = 50.0$. Since all models are trained from $0.0 \leq t \leq 25.0$, the latter is an additional test of time extrapolation.

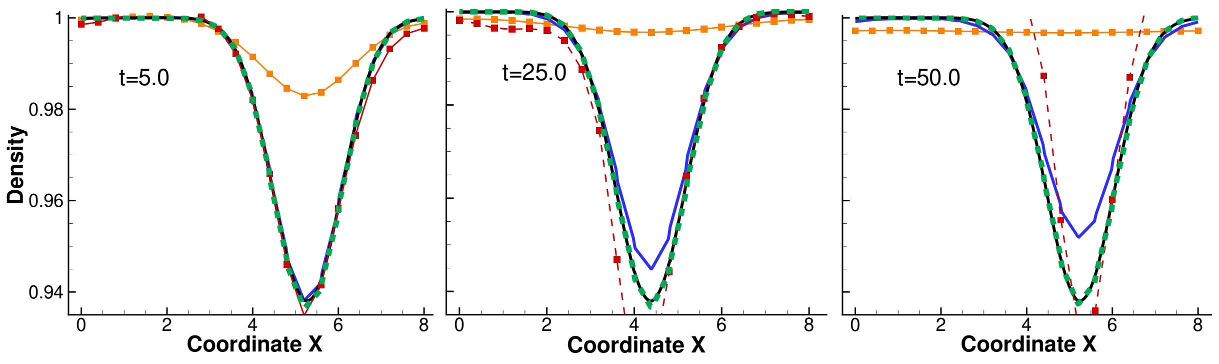
For the post-processing correction results given in Figure 6.13a, it is visible that improvements are found within the training time for $p = 0$. The corrected $p = 0$ solution at $t = 5.0$ approximates the density values of $p = 2$, with slight deviations close to the vortex core. The corrected solution at later time $t = 25.0$ only approximates the $p = 1$ simulation results, which is nevertheless an increase in accuracy of one polynomial degree. As for the extrapolation in time, the model is not capable of predicting a smooth correction, such that the density exhibits significant fluctuations and deviates from any solution. Looking at the $p = 1$ post-processing corrections, it is first of all visible that this discretization is missing higher frequencies. This results in a staggered solution rather than the smooth curve obtained with $p = 2$. Nevertheless, the correction approximates the higher order solution within the training time up to $t = 25.0$, such that no dissipation of the vortex occurs. No significant improvements are visible at later time $t = 50.0$, and the corrected curve only slightly increases the density at the vortex core.

Looking at Figure 6.13b, which shows the results of the iterative correction model, it becomes evident that the model trained to correct $p = 0$ deviates early from the expected corrections. This error grows in time, such that the ML augmented vortex at $t = 50.0$ diverges from the expected $p = 2$ simulation result. Similarly to the post-processing correction, the iterative correction model displays better predictive capabilities for $p = 1$ than for $p = 0$. Up to $t = 50.0$, the $p = 1$ correction demonstrates increased accuracy, producing solutions that closely resemble the simulation results obtained with one polynomial degree increase, i.e. $p = 2$.

Figures 6.14 and 6.15 depict quantitative results, in this case the MAE for each sample at different times for both the post-processing and the iterative correction,



(a) Post-processing correction with supervised learning



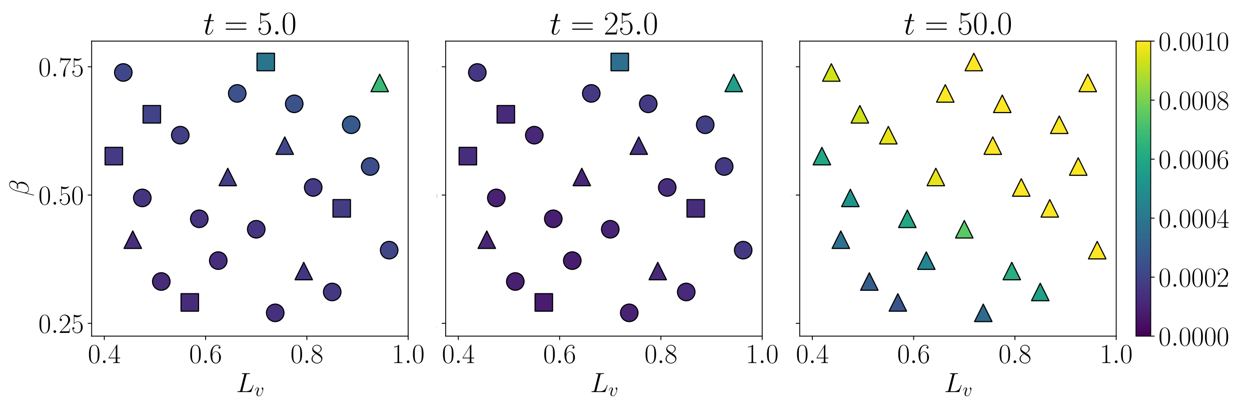
(b) Iterative correction with supervised learning and noise regularization

Figure 6.13: Density plotted along the middle of the z-axis for simulations with polynomial degrees $p \in \{0, 1, 2\}$ and their corresponding ML corrections at times $t \in \{5.0, 25.0, 50.0\}$ for test set with $\beta = 0.7185$ and $L_v = 0.9438$

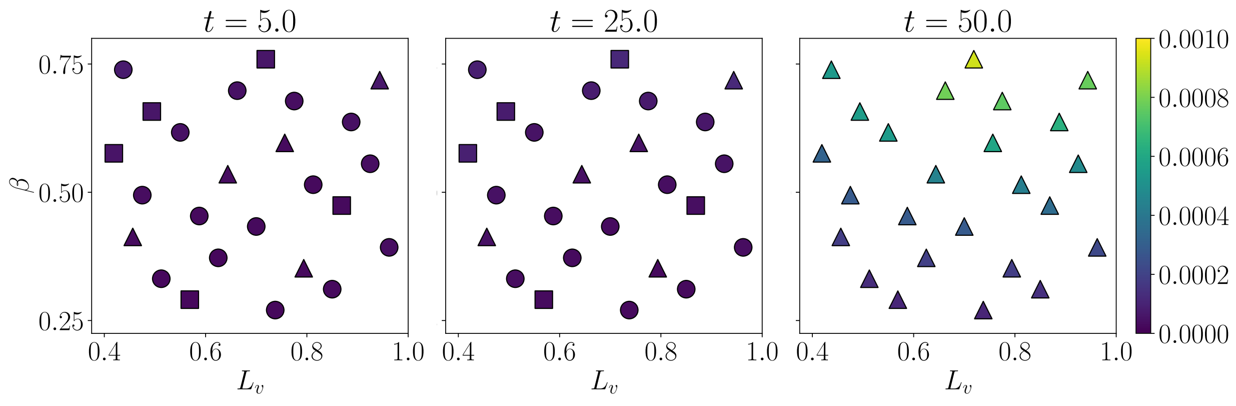
respectively. Circles represent training samples, squares validation samples, and diamonds test samples. Here, only the resulting MAE for the density variable ρ is shown, but similar trends are observed for the other corrected variables. Considering the post-processing corrections in Figure 6.14, the trend that the ML model is not capable to predict accurate corrections beyond the training time is again confirmed by the results at $t = 50$. This is presented for both $p = 0$ and $p = 1$ in Figures 6.14a and 6.14b, and especially pronounced for increased vortex strengths and vortex lengths. For the corrections within the training time $0.0 \leq t \leq 25.0$, the models are capable to predict highly accurate results for $p = 1$ across all samples and all variables. For $p = 0$, the training samples show increased errors at the boundary of the design space where high β and L_v are encountered. Not surprisingly, the maximum error is found for the test sample with greatest vortex strength and length, for which the density was previously plotted in Figure 6.13.

For the iterative correction of $p = 0$, given in Figure 6.15a, increased MAE values are found early at $t = 5.0$, especially for greater vortex strengths. These errors grow over time, as can be expected due to the autoregressive nature of the iterative correction, leading to the poorest results. Thus, in this case, training under noise did not lead to a sufficient regularization to achieve stable long-term predictions. The overall best performance is found for he iterative correction for $p = 1$, given in Figure 6.15b. Here, the model which is trained under noise is also capable to extrapolate in time, leading to excellent results across all samples after 100 time steps at $t = 50.0$.

In general, the plots indicate that the $p = 0$ correction is more difficult to learn for both post-processing and iterative correction approaches. Better performance for the $p = 1$ correction might have several reasons: first of all, the error grows slower, decreasing the variance in the dataset for the post-processing correction and leading to a slower grow of the ML model error in the iterative correction approach. Secondly, $p = 1$ for a 2D problem with four equations leads to 12 degrees of freedom per element. Thus, employing first and second derivatives as feature inputs, this results in three times as many inputs as for $p = 0$. Although this also increases the number of outputs to be predicted, the additional feature information might lead to improved results and an overall best performance for the iterative correction for $p = 1$.

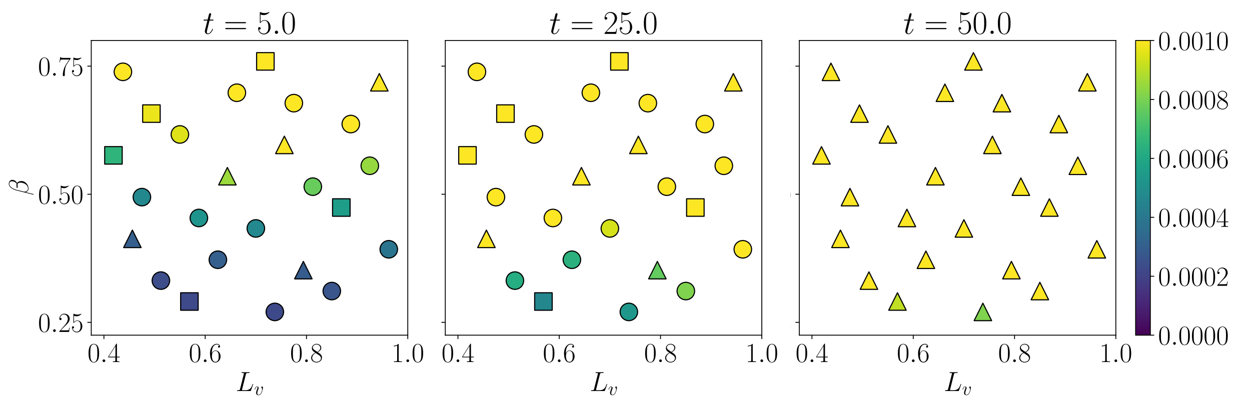


(a) Correction of polynomial degree $p = 0$

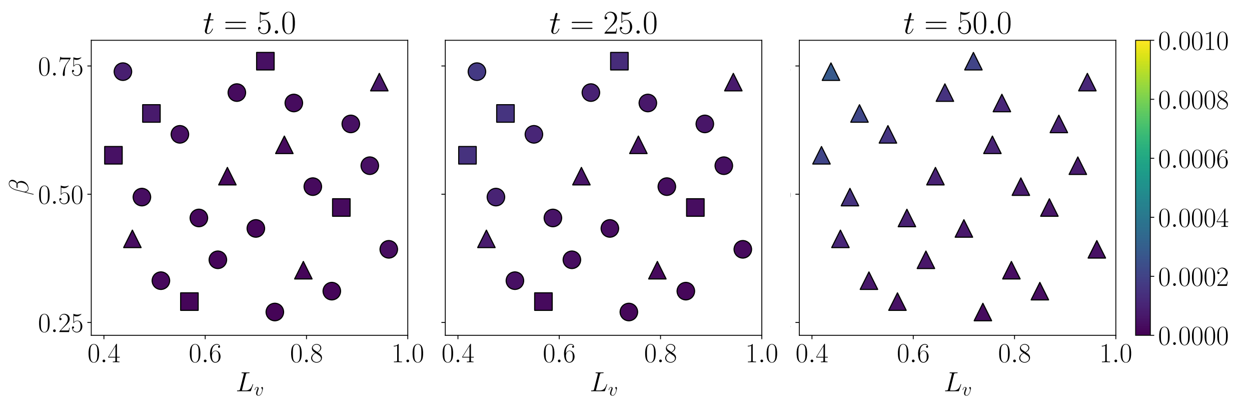


(b) Correction of polynomial degree $p = 1$

Figure 6.14: MAE for corrected ρ across all samples for post-processing correction with supervised learning. Note that training was conducted from $0.0 \leq t \leq 25.0$, for $\Delta t = 0.5$. Legend: ○ training samples, □ validation samples, △ test samples.



(a) Correction of polynomial degree $p = 0$



(b) Correction of polynomial degree $p = 1$

Figure 6.15: MAE across all samples for iterative correction with supervised learning and noise regularization. Note that training was conducted from $0.0 \leq t \leq 25.0$, for $\Delta t = 0.5$. Legend: ○ training samples, □ validation samples, △ test samples.

6.3 Concluding Remarks

This chapter investigated ML based corrections to increase the accuracy of low-order unsteady DG simulations. Three different correction and ML training combinations were compared: firstly, a methodology decoupled from the CFD solver with a model trained under supervised learning for a post-processing correction. Beyond the training time, the resulting corrections show poor long-term predictive capabilities. Nevertheless, this method is the most reliable correction approach within the training time window across all samples, achieving accuracy comparable to solutions of at least one polynomial degree higher. Another advantage of this decoupled approach is its simplicity in terms of data generation, training, and correction application - due to the clear separation between ML model and CFD solver.

Secondly, with the aim to improve predictions beyond the training time, an iterative correction approach trained under supervised learning was suggested. On the 1D linear advection problem, this led to an increased accuracy at later time steps, and adding regularization in form of noise increased generalization capabilities. For the 2D test case, the noise regularization approach shows the best corrective capabilities for $p = 1$, but poor results for polynomial degree $p = 0$. In general, it is concluded that this approach is not reliable, since the fundamental limitation of autoregressive models is the risk of error accumulations. This ultimately renders an otherwise robust CFD solver over long-term horizons unstable.

Finally, to decrease autoregressive errors, RL was explored as an alternative to adjoint-driven training or differentiable solvers, for coupling not only the correction, but also the model training with the solver. While RL enables interactions with the CFD solver, allowing to incorporate the model error during training, the resulting corrections for the 1D test case are not sufficient. Requiring up to 1'889 CFD simulations for each of the 20 training samples, the training is significantly more expensive compared to the supervised learning approaches. Additionally, the correction performance is the poorest among all methods. PPO is known to be sample inefficient, such that other RL algorithms with improved sample efficiency could be considered, for example the the Deep Deterministic Policy Gradient (DDPG) algorithm [76]. However, this choice comes at the expense of reduced training stability compared to PPO. Other aspects that could be revisited include adjusting the level of exploration, the shape of the reward function, and including a validation dataset - although improvements based on the latter might be marginal, since it was shown that the model already performs poorly on training samples. Overall, it can be stated that if dense training data is available and if the state of the environment under consideration is high-dimensional, such as in this case finding a correction for each element in the discretized domain at every time step, the PPO algorithm is currently not a viable option. These findings suggest that the here proposed online approach using PPO is not a feasible substitute for differentiable solvers.

In summary, this chapter has shown that ML corrections of unsteady low-order DG simulations are feasible but have in general several limitations. Using the decoupled method of a post-processing correction with supervised learning can be an attractive surrogate model within a constrained design space and confined training time windows. For future work, the method could be extended to more complex flows, including non-periodic, 3D, and high Reynolds number cases. As for the aim of predicting long-term corrections beyond the training time with an iterative approach, it can be concluded that the methods explored in this work coupling ML model and CFD solver do not yet demonstrate sufficient reliability to justify further investigations.

Chapter 7

Conclusion and Outlook

To conclude this work, the main research question introduced in section 1.3 is repeated here.

Main research question
Can ML methods be employed to increase the accuracy of inexpensive low-fidelity CFD simulations?

To investigate this question, correction methods were developed and tested across three different scenarios: the correction of coarse grid steady FV simulations, low-order steady DG simulations, and low-order unsteady DG simulations. The effectiveness of the proposed approaches has been demonstrated throughout this work. For steady coarse grid FV simulations, corrections were tested on turbulent flows around the RAE2822 airfoil and the LANN wing, employing the RANS equations and the negative version of the Spalart-Allmaras turbulence model. For steady DG simulations, the same airfoil geometry was used, and for the 3D case, the laminar flow around a delta wing was investigated. For unsteady problems, the proposed correction methods were first applied to a simple 1D linear advection of a sine curve. Subsequently, the two most promising approaches were further tested on the convection of an isentropic vortex described by the Euler equations.

The results have shown that the proposed methods can be leveraged as effective surrogate models accurately reproducing certain values, including lift and pressure coefficient, or reconstructing phenomena such as vortex shedding or convection in time. Thus, this work can be leveraged in many-query problems, where a multitude of parametrized simulations within a defined design space have to be conducted.

In the following, the three sub-questions, which explore robustness, accuracy, and efficiency of the proposed methods, are addressed to further elaborate on the main research question in more detail. The advantages and disadvantages of the proposed methods are discussed, and potential future research directions are proposed.

7.1 Review of Results and Research Questions

Question 1

Can ML models *robustly* infer corrections for low-fidelity simulations under varying conditions?

The first question asks if a consistent pattern exists between low-fidelity flow field features and the discretization error, which can be learned by data-driven models. This work has shown that ML models are powerful function approximators for the inference of a point-wise correction for low-fidelity simulations. Thus, the trained models have to a certain extent the potential to effectively find a relationship between a low-fidelity simulation output and associated errors derived from high-fidelity simulations.

The errors to be learned are related to the first and second derivatives of the flow variables to be corrected or the polynomial coefficients, which are effectively used as feature inputs to the ML models. This is in line with classical approaches, as derivatives are often utilized as error indicators for adaptive refinement techniques. Derivatives as error indicators are a natural choice, since it can be assumed that steep gradients, encountered for example at shock positions, are located in areas of high resolution of the respective discretization. Additionally, using derivatives includes information not only from the local element, but also from neighboring ones, which improves point-wise predictions significantly. This observation resonates with the finding from the FV study, in which the GNN demonstrates superior corrective capabilities compared to the NN and RF models, especially in cases with discontinuities. Employing a GNN enables to efficiently collect and aggregate feature information from neighboring elements, resulting in a richer feature representation to effectively infer an element-wise correction. If boundary values are of interest, such as the surface pressure or lift coefficient, employing a local cell Reynolds number is of relevance, as it allows to embed the wall distance as an input value of the models. Other values, such as global ones, like the Mach number and angle of attack, or grid related values, including skewness and cell volume, have not shown any significant influence. Nevertheless, this finding does not seem to be generally valid, as for the considered unsteady 1D linear advection problem the use of global values, here the amplitude and constant velocity coefficient, resulted in major improvements, whereas for the unsteady problem in which the convection of a vortex was studied, including the global features of vortex strength and vortex length, did not result in any relevant performance boost.

As for any data-driven methodology, the extent to which the learned patterns can generalize within the design space and to out of distribution data is of importance. The robustness of the proposed methodology was assessed in different settings. Improved low-fidelity solutions are encountered for test samples within the training design space, which means that the trained models succeed in interpolation tasks. Difficulties and decreased performance for all models are nevertheless found for problems with discontinuities, specifically for shocks. Here, the RF model shows inferior performance compared to deep learning models, i.e. the NN and GNN, a finding which aligns with previous work [74, 144]. Examining extrapolation capabilities of the trained models was done in different ways. First, the GNN model trained on coarse grid CFD simulations for the RAE2822 was applied to a slightly altered geometry, namely the RAE5212 airfoil. Here, a tremendous drop of performance was found. Secondly, during the unsteady investigations, test samples at the boundary of the design space not surrounded by training and validation samples can be seen as extrapolation test, although to a lesser extent than the application to a different geometry. Similarly to difficulties encountered at high angles of attack, increased errors are encountered for high amplitudes and velocity coefficients for the 1D linear advection, as well as for high vortex strengths and lengths for convection of a 2D vortex. Additionally, the models trained under a supervised learning method showed the most robust and accurate inferences, while an online learning approach based on reinforcement learning did not increase the generalization capabilities within the design space. The final and most intricate extrapolation test was conducted with respect to time, assessing if the ML models are capable of predicting corrections beyond their training time. Models trained for a post-processing correction naturally fail here, as the error between low- and high-fidelity trajectories diverge over time, leading to out of distribution data which is simply not encountered during the training time. Enforcing an even distribution of the correction along the time dimension by applying an iterative correction approach lessens this issue, at the cost of introducing autoregressive errors. Introducing regularization in form of noise resulted in improved long-term performance for many cases, while the investigated online learning approach, namely the PPO reinforcement learning algorithm, did not increase the robustness in time. In general, none of the proposed methods show satisfactory reliability in terms of extrapolation in time.

It was also assessed whether the model hyperparameters, which alongside the model parameters define the predictions, can be transferred and reused to learn new tasks. For both the FV and DG study, it was found that the problem could be effectively scaled from 2D to 3D. Most of the RF and GNN hyperparameters were reused from the 2D test cases, resulting in comparable performance, while only changing for the GNN the learning rate and the number of epochs, as well as the number of decision trees for the RF. This highlights the potential for transferring hyperparameters to scale problems within the same discretization scheme. However, although the results of this were not shown here but presented in [85], applying the same GNN hyperparameters to a different grid and discretization method, specifically from FV to the DG scheme, under identical conditions for the RAE2822 airfoil, a noticeable drop in performance was observed. This suggests that while GNN hyperparameters may generalize well within the same discretization, they are less transferable across fundamentally different discretization methods. In contrast, the RF model, which is influenced by fewer hyperparameters compared to NNs and GNNs, exhibited consistent performance when the hyperparameters found during the 2D coarse grid FV study were applied to the 2D DG test case [85]. This robustness can be attributed to the RF model's lower sensitivity to hyperparameter choices, as its training relies mostly on ensemble averaging of weak learners and less on architecture-specific tuning, as is the case for deep learning models. The observed sensitivity of the GNN and the robustness of the RF underline the importance of model choice and problem specific hyperparameter optimization.

Question 2

Can the corrected solutions *accurately* approximate the respective high-fidelity ones?

The second question deals with the results achieved with respect to the accuracy of the ML augmented low-fidelity simulations. Significant improvements have been achieved in terms of the corrected flow fields, leading to the accurate reconstruction of vortex shedding, as well as enhanced surface pressure and lift coefficients. For example, within the design space and training time, accuracy of at least one polynomial degree higher was achieved for the convection of a vortex. Thus, the proposed methods can be employed as a potential surrogate model, resulting in accurate and fast predictions for such values.

Limitations of the achievable accuracy are not only posed by extreme cases, such as shocks or samples at the border of the design space, which require certain extrapolation capabilities. In the proposed methods, the upper limit of accuracy is ultimately defined by either the coarse grid or the low-order polynomial, as well as by the transfer operator, which maps the high-fidelity solution to the low-fidelity discretization. In the investigated cases, while the pressure and lift coefficients were largely unaffected, other aerodynamic values of interest influenced by velocity gradients showed reduced accuracy. This was presented in the FV study, in which the derived skin friction coefficients from the ML corrected solutions closely match those derived from the mapped solution, showing significant improvements over the low-fidelity discretization. However, the achieved accuracy does not reach the precision of the fine grid solution, since the boundary layer elements are generally too coarse to capture accurate gradients.

Additionally, the accuracy is impacted not only by discretization and mapping limitations but also by errors introduced by the ML models themselves. These inaccuracies are especially evident in regions with shocks, despite the ML corrections significantly improving the respective low-fidelity solutions. Some ML models showed oscillatory behavior in derived properties like the pressure coefficient. For time-dependent problems, the contribution of the ML error becomes even more pronounced. On the one hand, the model trained for a post-processing correction showed high accuracy for interpolation tasks within the design space and the training time, while an accuracy decrease is obtained beyond its training time, rendering the model incapable of extrapolation in time. On the other hand, while an autoregressive approach with an iterative correction in-between solver iterations promises to extend stability over longer time periods, its accuracy diminishes due to the accumulation of prediction errors over successive steps. This error propagation poses a fundamental limitation, rendering the correction approach unreliable. As the predictions at each successive time step rely on the outputs of previous steps, minor inaccuracies can amplify, leading to significant deviations from the true solution over extended periods.

Question 3

Can all of the above be done *efficiently* to maintain the low cost of the low-fidelity simulation?

The last question is concerned with the efficiency of the proposed methods. All correction approaches rely on a point-wise inference, in the considered cases vertex-wise predictions for the FV scheme and element-wise predictions for the DG discretization. Correcting each vertex or element separately allows to obtain more training instances from a single simulation and to apply the trained model on new datasets, since it is not bound to the number of points in the grid. It was also shown that the steady correction methods are scalable from 2D to 3D, while the training cost increases with the number of training instances and feature inputs. Nevertheless, training an ML model is a one-time investment, after which it can infer corrections within seconds. For example, considering the DG correction for the 3D delta wing, the $p = 0$ simulation, which takes an average of 1 minute to compute, can be corrected to approximate the pressure field and vortex shedding obtained from the $p = 2$ simulation, which takes over 7 hours on average.

Additionally, employing the trained ML model only on low-fidelity discretizations with few degree of freedom ensures that computational demands remain relatively low. This, since only the low-fidelity simulations have to be computed, oftentimes feasible without HPC access. Additionally, depending on the complexity of the ML model architecture, memory requirements can be kept low. Although the training of GNN models are significantly sped up on GPUs, deploying the trained model does not necessitate GPU hardware.

The major cost and challenge for the proposed methods stem from generating high-fidelity data and the need to conduct resource intensive hyperparameter optimizations. However, in practical industrial applications, extensive datasets may already exist due to years of accumulated simulation expertise in the respective field. Leveraging such high-fidelity data could enable the training of ML models to learn and correct discretization errors, although certain aspects of the methods, such as the injection for the FV discretization, would need to be revisited. Nevertheless, using historical data could reduce the need for additional costly simulations, as low-fidelity simulations together with data-driven corrections can be used to quickly sample new solutions.

As for the comparison between supervised and reinforcement learning approaches, the latter promises higher flexibility due to its reward function. Using the PPO algorithm, the stochastic nature of training RL agents needing to explore the interaction with its environment comes along with the disadvantage of high simulation cost. For the investigated problem, the supervised learning approaches were deemed significantly more efficient than the RL method. As a conclusion, it can be suggested to focus on supervised learning paradigms if the data allows to, especially for high-dimensional problems.

7.2 Future Potential

In light of these conclusions, is there a way to improve the robustness, accuracy, and efficiency of the proposed methods? Ultimately, the interplay between these three traits involves a trade-off: employing ML methods for CFD tends to reduce accuracy while shifting the computational cost to data generation and model training. Although deploying the trained ML model is relatively efficient, its generalization is confined within the training design space, and coupling correction with solver iterations negatively affect the robustness of the CFD solver, ultimately introducing stability and reliability issues.

Robustness can be discussed in terms of ML generalization capabilities and in terms of the reduction of the stability when coupling ML methods with the CFD solver. As for the first point, increasing the training design space, for example by including several relevant geometries, will increase the range of applicability range of any ML model. Furthermore, the proposed methods of this work are purely data-driven, and a worthwhile further investigation is the use of physics-informed aspects. This could involve for example relevant values such as resulting lift or drag coefficient in a loss function, or imposing the conservation laws as well as the boundary conditions of the problem during the training of the ML model, by employing positivity preserving schemes. Using physics-informed methods does not only potentially improve generalization capabilities but might also increase the stability when coupling ML and CFD methods, as a physics-informed loss acts as regularization, possibly reducing ML prediction errors. For the correction of unsteady simulations, reinforcement learning has not shown to be a feasible method. An alternative, investigated in other research work, is the use of adjoint driven frameworks. Although requiring modifications within the CFD solver, this might proof to be a more robust approach to couple ML and CFD methods.

To increase the accuracy of the proposed methods for relevant aerodynamic applications, the low-fidelity discretization needs to be adjusted close to the boundary layer, where relevant values such as the drag coefficient are evaluated. For coarse grids, the boundary layer can be refined, leading to anisotropic boundary elements capturing velocity gradients more accurately. Such a grid refinement can also be applied to the correction of DG solutions with low polynomial degrees, although here the possibility of retaining higher order close to the airfoil might also be considered. Alternatively, the classical approach of using wall functions could be used to capture the high-fidelity velocity gradients on low-fidelity discretizations.

To decrease the cost of the proposed methods, the main limiting factors, namely the dataset generation and model training, need to be addressed. Oftentimes, ML methods are applied with the justification that there is an abundance of simulation and experimental data available. Unfortunately, this abundance of data for aerodynamic applications is usually not publicly available, making it necessary to generate costly high-fidelity data from scratch, as was done for this work. Thus, to make ML for CFD methods more accessible and efficient, an effort is needed to create available and standardized datasets. To accelerate model training, a possible suggestion is to not only reuse previously found hyperparameters but also provide pre-trained models, which can subsequently be fine-tuned for specific tasks. These two suggestions, providing generic datasets as well as using pre-trained models, have lead to the success of large language models in the field of natural language processing, and a similar effort is needed to make ML methods successful for CFD applications.

Finally, it is important to reflect on how the proposed approaches compare to or complement already existing methods. The approaches in this thesis could serve as viable surrogate models, enabling fast many-query sampling of CFD solutions on low-fidelity discretizations. Evaluating not only surface and integral values but also the field variables allows for more insights, making these methods an attractive alternative to established surrogate models, which often only predict surface related variables.

Additionally, the proposed correction framework could complement classical numerical methods which utilize low-fidelity discretizations, particularly multigrid or adaptive refinement techniques, presented in chapter 1. In the context of multigrid methods, ML predicted corrections could be used at intermediate levels of the multigrid cycle, similarly to [102], potentially accelerating convergence. The benefit of using ML predictions in this manner is to serve only as guidance to enhance the convergence rate, while the final accuracy is ultimately dictated by the discretization of the finest level. This ensures the reliability of the overall solution - of course only if sufficient convergence is achievable. Similarly, for adaptive refinement methods, the ML predictions could be utilized not as correction term but as an error indicator. By identifying regions of high discretization error, these predictions could guide the refinement of the mesh or the polynomial degree more effectively.

Identifying such areas where the ML model would mainly act as a guidance, while the preservation of the conservation laws is still ensured by the CFD solver, offers a promising direction for future advancements of ML for CFD.

Bibliography

[1] Advisory Group for Aerospace Research and Development (AGARD). AGARD Advisory Report No. 138 - Experimental Data Base for Computer Program Assessment. Technical Report AGARD-AR-138, North Atlantic Treaty Organization, Fluid Dynamics Panel, 1979.

[2] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A Next-Generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631. Association for Computing Machinery, 2019.

[3] S. Allmaras, F. Johnson, and P. Spalart. Modifications and Clarifications for the Implementation of the Spalart-Allmaras Turbulence Model. In *Seventh International Conference on Computational Fluid Dynamics (ICCFD7)*, 2012.

[4] R. Amit, R. Meir, and K. Ciosek. Discount Factor as a Regularizer in Reinforcement Learning. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 269–278. PMLR, 2020.

[5] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem. What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study, 2020. *arXiv:2006.05990*.

[6] S. M. Anwar, M. Majid, A. Qayyum, M. Awais, M. Alnowami, and M. K. Khan. Medical Image Analysis using Convolutional Neural Networks: A Review. *Journal of Medical Systems*, 42(11):226, 2018.

[7] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner. Learning Data-Driven Discretizations for Partial Differential Equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019.

[8] F. Bassi, L. Botti, A. Colombo, and S. Rebay. Agglomeration Based Discontinuous Galerkin Discretization of the Euler and Navier–Stokes Equations. *Computers & Fluids*, 61:77–85, 2012.

[9] F. Bassi and S. Rebay. A High-Order Accurate Discontinuous Finite Element Method for the Numerical Solution of the Compressible Navier–Stokes Equations. *Journal of Computational Physics*, 131(2):267–279, 1997.

[10] F. Bassi and S. Rebay. GMRES Discontinuous Galerkin Solution of the Compressible Navier-Stokes Equations. In *Discontinuous Galerkin Methods*, pages 197–208. Springer Berlin Heidelberg, 2000.

[11] A. Beck and M. Kurz. A Perspective on Machine Learning Methods in Turbulence Modeling. *GAMM-Mitteilungen*, 44(1), 2021.

[12] A. Beck and M. Kurz. Toward Discretization-Consistent Closure Schemes for Large Eddy Simulation using Reinforcement Learning. *Physics of Fluids*, 35(12), 2023.

[13] P. Bekemeyer, A. Bertram, D. A. H. Chaves, M. D. Ribeiro, A. Garbo, A. Kiener, C. Sabater, M. Stradtner, S. Wassing, M. Widhalm, S. Goertz, F. Jaeckel, R. Hoppe, and N. Hoffmann. *Data-Driven Aerodynamic Modeling Using the DLR SMARTy Toolbox.* AIAA AVIATION 2022 Forum, 2022. AIAA 2022-3899.

[14] R. E. Bellman. A Markov Decision Process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[15] R. E. Bellman. *Dynamic Programming*, volume 33. Princeton University Press, Princeton, 1957.

[16] M. J. Berger and J. Oliger. Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. *Journal of Computational Physics*, 53(3):484–512, 1984.

[17] O. Bidar, P. He, S. Anderson, and N. Qin. *Aerodynamic Shape Optimisation Using a Machine Learning-Augmented Turbulence Model.* AIAA SCITECH 2024 Forum, 2024. AIAA 2024-1231.

[18] C. Bishop. *Pattern Recognition and Machine Learning*, pages XX–778. Springer New York, first edition, 2006.

[19] J. Blazek. *Computational Fluid Dynamics: Principles and Applications.* Butterworth-Heinemann, third edition edition, 2005.

[20] S. Boblest, F. Hempert, M. Hoffmann, P. Offenhäuser, M. Sonntag, F. Sadlo, C. W. Glass, C.-D. Munz, T. Ertl, and U. Iben. Toward a Discontinuous Galerkin Fluid Dynamics Framework for Industrial Applications. In *High Performance Computing in Science and Engineering '15*, pages 531–545. Springer International Publishing, 2016.

[21] R. Borrell, D. Dosimont, M. Garcia-Gasulla, G. Houzeaux, O. Lehmkuhl, V. Mehta, H. Owen, M. Vázquez, and G. Oyarzun. Heterogeneous

CPU/GPU Co-Execution of CFD Simulations on the POWER9 Architecture: Application to Airplane Aerodynamics. *Future Generation Computer Systems*, 107:31–48, 2020.

[22] X. Bouthillier, C. Laurent, and P. Vincent. Unreproducible Research is Reproducible. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 725–734. PMLR, 2019.

[23] A. Brandt. Multi-Level Adaptive Solutions to Boundary-Value Problems. *Mathematics of Computation*, 31:333–390, 1977.

[24] M. Brazell, B. R. Ahrabi, and D. Mavriplis. *Discontinuous Galerkin Turbulent Flow Simulations of NASA Turbulence Model Validation Cases and High Lift Prediction Workshop Test Case DLR-F11.* 54th AIAA Aerospace Sciences Meeting, 2016. AIAA 2016-0861.

[25] L. Breiman. Random Forests. *Machine Learning*, 45:5–32, 2001.

[26] X. Bresson and T. Laurent. Residual Gated Graph ConvNets, 2018. *arXiv:1711.07553.*

[27] S. L. Brunton, B. R. Noack, and P. Koumoutsakos. Machine Learning for Fluid Mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.

[28] N. K. Burgess and D. J. Mavriplis. High-Order Discontinuous Galerkin Methods for Turbulent High-Lift Flows. In *Seventh International Conference on Computational Fluid Dynamics (ICCFD7)*, 2012.

[29] G. Calzolari and W. Liu. Deep Learning to Replace, Improve, or Aid CFD Analysis in Built Environment Applications: A Review. *Building and Environment*, 206, 2021.

[30] F. J. Cantarero-Rivera, R. Yang, H. Li, H. Qi, and J. Chen. An Artificial Neural Network-Based Machine Learning Approach to Correct Coarse-Mesh-Induced Error in Computational Fluid Dynamics Modeling of Cell Culture Bioreactor. *Food and Bioproducts Processing*, 143:128–142, 2024.

[31] N. Ceresola, L. Djayapertapa, R. Heinrich, S. Leichner, T. E. Berglind, N. Caballero Rubiato, B. Paranta, and V. Brunet. Task 5 Oscillating LANN Wing. *GARTEUR AD(AG38) - Time Accurate Methods, Chapter 7*, 2006.

[32] J. Chen, E. Hachem, and J. Viquerat. Graph Neural Networks for Laminar Flow Prediction around Random Two-Dimensional Shapes. *Physics of Fluids*, 33, 2021.

[33] Q. Chen and W. Xu. A Zero-Equation Turbulence Model for Indoor Airflow Simulation. *Energy and Buildings*, 28(2):137–144, 1998.

[34] X.-W. Chen and X. Lin. Big Data Deep Learning: Challenges and Perspectives. *IEEE Access*, 2:514–525, 2014.

[35] F. Chollet. *Deep Learning with Python.* Manning, second edition, 2021.

[36] B. Cockburn and C.-W. Shu. Runge-Kutta Discontinuous Galerkin Methods for Convection-Dominated Problems. *Journal of Scientific Computing*, 16:173–261, 2001.

[37] P. Cook, M. McDonald, and M. Firmin. Aerofoil RAE 2822 - Pressure Distributions, and Boundary Layer and Wake Measurements. Technical report, Experimental Data Base for Computer Program Assessment, AGARD Report AR 138, 1979.

[38] J. Cornell and R. Berger. Factors that Influence the Value of the Coefficient of Determination in Simple Linear and Nonlinear Regression Models. *Phytopathology*, 77:63–70, 1987.

[39] P. V. Coveney and R. R. Highfield. When we can Trust Computers (and when we can't). *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2197), 2021.

[40] T. Davydzenka and P. Tahmasebi. High-Resolution Fluid–Particle Interactions: a Machine Learning Approach. *Journal of Fluid Mechanics*, 938, 2022.

[41] F. De Avila Belbute-Peres, T. Economon, and Z. Kolter. Combining Differentiable PDE Solvers and Graph Neural Networks for Fluid Flow Prediction. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 2402–2411. PMLR, 2020.

[42] F. M. de Lara and E. Ferrer. Accelerating High Order Discontinuous Galerkin Solvers using Neural Networks: 1D Burgers' Equation. *Computers & Fluids*, 235, 2022.

[43] R. Deiterding. Adaptive Mesh Refinement - Theory and Applications. *Construction and Application of An AMR Algorithm for Distributed Memory Computers*, pages XIV–554, 2005.

[44] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. OpenAI Baselines, 2017. *GitHub repository.*

[45] K. Do, T. Tran, and S. Venkatesh. Graph Transformation Policy Network for Chemical Reaction Prediction. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 750–760. Association for Computing Machinery, 2019.

[46] K. Duraisamy. Perspectives on Machine Learning-Augmented Reynolds-Averaged and Large Eddy Simulation Models of Turbulence. *Phys. Rev. Fluids*, 6, 2021.

[47] K. Duraisamy, G. Iaccarino, and H. Xiao. Turbulence Modeling in the Age of Data. *Annual Review of Fluid Mechanics*, 51:357–377, 2019.

[48] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. Implementation Matters in Deep RL: A Case Study on PPO and TRPO. In *International Conference on Learning Representations*, 2020.

[49] C. Fallet, A. Garbo, A. Kiener, and P. Bekemeyer. *Reduced Order Model for Steady Aerodynamics Applications Based on Navier-Stokes Residual Vector Minimization.* AIAA AVIATION 2023 Forum, 2023. AIAA 2023-3715.

[50] E. Ferrer, G. Rubio, G. Ntoukas, W. Laskowski, O. Mariño, S. Colombo, A. Mateo-Gabín, H. Marbona, F. Manrique de Lara, D. Huergo, J. Manzanero, A. Rueda-Ramírez, D. Kopriva, and E. Valero. HORSES3D: A High-Order Discontinuous Galerkin Solver for Flow Simulations and Multi-Physics Applications. *Computer Physics Communications*, 287, 2023.

[51] A. Ferrero, A. Iollo, F. Larocca, M. Loffredo, and E. Menegatti. Field Inversion and Machine Learning Strategies for Improving RANS Modelling in Turbomachinery. In *14th European Conference on Turbomachinery Fluid dynamics & Thermodynamics*, 2021.

[52] J. H. Ferziger, M. Perić, and R. L. Street. *Computational Methods for Fluid Dynamics.* Springer Berlin, Heidelberg, third edition, 2019.

[53] M. Fey and J. E. Lenssen. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[54] G. C. Firth. LANN Wing Design. Technical report, Lockheed-Georgia Co., 1983.

[55] M. Fogleman, J. Lumley, D. Rempfer, and D. Haworth. Application of the Proper Orthogonal Decomposition to Datasets of Internal Combustion Engine Flows. *Journal of Turbulence*, 5, 2004.

[56] H. Fujiyoshi, T. Hirakawa, and T. Yamashita. Deep Learning-Based Image Recognition for Autonomous Driving. *IATSS Research*, 43(4):244–252, 2019.

[57] K. Fukami, K. Fukagata, and K. Taira. Super-Resolution Reconstruction of Turbulent Flows with Machine Learning. *Journal of Fluid Mechanics*, 870:106–120, 2019.

[58] K. Fukami, K. Fukagata, and K. Taira. Assessment of Supervised Machine Learning Methods for Fluid Flows. *Theoretical and Computational Fluid Dynamics*, 34:497–519, 2020.

[59] K. Fukami, K. Fukagata, and K. Taira. Super-Resolution Analysis via Machine Learning: A Survey for Fluid Flows. *Theoretical and Computational Fluid Dynamics*, pages 421–444, June 2023.

[60] H. Gao, L. Sun, and J.-X. Wang. Super-Resolution and Denoising of Fluid Flow using Physics-Informed Convolutional Neural Networks without High-Resolution Labels. *Physics of Fluids*, 33(7), 2021.

[61] 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, 2021.

[62] H. Ghraieb, J. Viquerat, A. Larcher, P. Meliga, and E. Hachem. Single-Step Deep Reinforcement Learning for Two- and Three-Dimensional Optimal Shape Design. *AIP Advances*, 12(8), 2022.

[63] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media, Inc., 2nd edition, 2019.

[64] S. Gössling and S. Dolnicar. A Review of Air Travel Behavior and Climate Change. *WIREs Climate Change*, 14(1), 2023.

[65] W. Hackbusch. *Multi-Grid Methods and Applications*, volume 4. first edition, 1985.

[66] M. U. Hadi, q. a. tashi, R. Qureshi, A. Shah, a. muneer, M. Irfan, A. Zafar, M. B. Shaikh, N. Akhtar, J. Wu, and S. Mirjalili. Large Language Models: A Comprehensive Survey of its Applications, Challenges, Limitations, and Future Prospects. *TechRxiv*, 2023.

[67] G. V. Hamilton. A Study of Trial and Error Reactions in Mammals. *Journal of Animal Behavior*, 1:33–66, 1911.

[68] J. Hammond, N. Pepper, F. Montomoli, and V. Michelassi. Machine Learning Methods in CFD for Turbomachinery: A Review. *International Journal of Turbomachinery, Propulsion and Power*, 7(2), 2022.

[69] B. N. Hanna, N. T. Dinh, R. W. Youngblood, and I. A. Bolotnov. Machine-Learning Based Error Prediction Approach for Coarse-Grid Computational Fluid Dynamics (CG-CFD). *Progress in Nuclear Energy*, 118, 2020.

[70] R. Hartmann and P. Houston. An Optimal Order Interior Penalty Discontinuous Galerkin Discretization of the Compressible Navier–Stokes Equations. *Journal of Computational Physics*, 227(22):9670–9685, 2008.

[71] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer New York, 2 edition, 2009.

[72] X. He, Y. Wang, and J. Li. Flow Completion Network: Inferring the Fluid Dynamics from Incomplete Flow Information using Graph Neural Networks. *Physics of Fluids*, 34(8), 2022.

[73] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep Reinforcement Learning that Matters, 2019. *arXiv:1709.06560* .

[74] D. Hines and P. Bekemeyer. Graph Neural Networks for the Prediction of Aircraft Surface Pressure Distributions. *Aerospace Science and Technology*, 137, 2023.

[75] J. Ho and A. West. *Field Inversion and Machine Learning for Turbulence Modelling Applied to Three-Dimensional Separated Flows.* AIAA AVIATION 2021 FORUM, 2021. AIAA 2021-2903.

[76] Y. Hou, L. Liu, Q. Wei, X. Xu, and C. Chen. A Novel DDPG Method with Prioritized Experience Replay. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 316–321, 2017.

[77] M. Injadat, F. Salo, A. B. Nassif, A. Essex, and A. Shami. Bayesian Optimization with Machine Learning Algorithms Towards Anomaly Detection. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2018.

[78] T. Jaakkola, S. Singh, and M. Jordan. Reinforcement Learning Algorithm for Partially Observable Markov Decision Problems. In *Advances in Neural Information Processing Systems*, volume 7, 1994.

[79] X. Ju, S. Farrell, P. Calafiura, D. Murnane, Prabhat, L. Gray, T. Klijnsma, K. Pedro, G. Cerati, J. Kowalkowski, G. Perdue, P. Spentzouris, N. Tran, J.-R. Vlimant, A. Zlokapa, J. Pata, M. Spiropulu, S. An, A. Aurisano, V. Hewes, A. Tsaris, K. Terao, and T. Usher. Graph Neural Networks for Particle Reconstruction in High Energy Physics Detectors, 2020. *arXiv:2003.11603.*

[80] J. Jumper, R. Evans, A. Pritzel, et al. Highly Accurate Protein Structure Prediction with AlphaFold. *Nature*, 596:583–589, 2021.

[81] G. K. W. Kenway, C. A. Mader, P. He, and J. R. R. A. Martins. Effective Adjoint Approaches for Computational Fluid Dynamics. *Progress in Aerospace Sciences*, 110, 2019.

[82] R. Keppens, M. Nool, G. Tóth, and J. Goedbloed. Adaptive Mesh Refinement for Conservative Systems: Multi-Dimensional Efficiency Evaluation. *Computer Physics Communications*, 153(3):317–339, 2003.

[83] H. Keramati, F. Hamdullahpur, and M. Barzegari. Deep Reinforcement Learning for Heat Exchanger Shape Optimization. *International Journal of Heat and Mass Transfer*, 194, 2022.

[84] A. Kiener and P. Bekemeyer. *Correcting Unsteady Low Order Discontinuous Galerkin Simulations.* AIAA SCITECH 2025 Forum, 2025. AIAA 2025-2046.

[85] A. Kiener, P. Bekemeyer, and S. Langer. Towards Fast Aerodynamic Simulations with Machine Learning Corrections for Discretization Errors. In *Deutscher Luft- und Raumfahrtkongress 2024*, 2024.

[86] A. Kiener, S. Langer, and P. Bekemeyer. Data-Driven Correction of Coarse Grid CFD Simulations. *Computers & Fluids*, 264, 2023.

[87] E. J. Kim and R. J. Brunner. Star–Galaxy Classification using Deep Convolutional Neural Networks. *Monthly Notices of the Royal Astronomical Society*, 464:4463–4475, 2016.

[88] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization, 2017. *arXiv:1412.6980*.

[89] T. N. Kipf and M. Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*, 2017.

[90] E. Kita and N. Kamiya. Error Estimation and Adaptive Mesh Refinement in Boundary Element Method, an Overview. *Engineering Analysis with Boundary Elements*, 25(7):479–495, 2001.

[91] K.-E. Ko and K.-B. Sim. Deep Convolutional Framework for Abnormal Behavior Detection in a Smart Surveillance System. *Engineering Applications of Artificial Intelligence*, 67:226–234, 2018.

[92] R. Koch, M. Sanjosé, and S. Moreau. *Acoustic Investigation of the Transonic RAE 2822 Airfoil with Large-Eddy Simulation.* 28th AIAA/CEAS Aeroacoustics Conference, 2022. AIAA 2022-2816.

[93] D. Kochkov, J. Smith, A. Alieva, Q. Wang, M. Brenner, and S. Hoyer. Machine Learning–Accelerated Computational Fluid Dynamics. *Proceedings of the National Academy of Sciences*, 118(21), 2021.

[94] J. C. Kok. A High-Order Low-Dispersion Symmetry-Preserving Finite-Volume Method for Compressible Flow on Curvilinear Grids. *Journal of Computational Physics*, 228(18):6811–6832, 2009.

[95] A. N. Kolmogorov, V. Levin, J. C. R. Hunt, O. M. Phillips, and D. Williams. Dissipation of Energy in the Locally Isotropic Turbulence. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 434(1890):15–17, 1991.

[96] A. N. Kolmogorov, V. Levin, J. C. R. Hunt, O. M. Phillips, and D. Williams. The Local Structure of Turbulence in Incompressible Viscous Fluid for very Large Reynolds numbers. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 434(1890):9–13, 1991.

[97] N. Kroll. *ADIGMA - A European Project on the Development of Adaptive Higher-Order Variational Methods for Aerospace Applications.* 47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition, 2009. AIAA 2009-176.

[98] N. Kroll, S. Langer, and A. Schwöppe. *The DLR Flow Solver TAU - Status and Recent Algorithmic Developments.* 52nd Aerospace Sciences Meeting, 2014. AIAA 2014-0080.

[99] A. Kuhnle, M. Schaarschmidt, and K. Fricke. Tensorforce: A TensorFlow Library for Applied Reinforcement Rearning, 2017. *GitHub repository.*

[100] L. Kuncheva and C. Whitaker. Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy. *Machine Learning*, 51:181–207, 2003.

[101] M. Kurz, P. Offenhäuser, and A. Beck. Deep Reinforcement Learning for Turbulence Modeling in Large Eddy Simulations. *International Journal of Heat and Fluid Flow*, 99, 2023.

[102] N. Kärcher and M. Wallraff. Accelerating cfd solver computation time with reduced-order modeling in a multigrid environment. *International Journal for Numerical Methods in Fluids*, 93(2):462–480, 2021.

[103] S. Langer. Agglomeration Multigrid Methods with Implicit Runge–Kutta Smoothers Applied to Aerodynamic Simulations on Unstructured Grids. *Journal of Computational Physics*, 277:72–100, 2014.

[104] S. Langer. Preconditioned Newton Methods to Approximate Solutions of the Reynolds Averaged Navier-Stokes Equations. Technical report, DLR Institut für Aerodynamik und Strömungstechnik, 2018.

[105] S. Langer. An Initial Investigation of Solving RANS Equations in Combination with Two-Equation Turbulence Models. Technical report, DLR Institut für Aerodynamik und Strömungstechnik, 2019.

[106] S. Langer and A. Schwöppe. TAU Hyperflex Report. Technical report, DLR Institut für Aerodynamik und Strömungstechnik, 2022.

[107] S. Langer, A. Schwöppe, and N. Kroll. Investigation and Comparison of Implicit Smoothers Applied in Agglomeration Multigrid. *AIAA Journal*, 53:2080–2096, 2015.

[108] Y. LeCun, Y. Bengio, and G. Hinton. Deep Learning. *Nature*, 521:436–444, 2015.

[109] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 1989.

[110] C. Y. Lee and S. Cant. A Grid-Induced and Physics-Informed Machine Learning CFD Framework for Turbulent Flows. *Flow, Turbulence and Combustion*, 112:407–442, 2023.

[111] M. Lee and R. D. Moser. Direct Numerical Simulation of Turbulent Channel Flow up to $Re_{\tau} \approx 5200$. *Journal of Fluid Mechanics*, 11(4):943–945, 2015.

[112] T. Leicht and R. Hartmann. Error Estimation and Anisotropic Mesh Refinement for 3D Laminar Aerodynamic Flow Simulations. *Journal of Computational Physics*, 229(19):7344–7360, 2010.

[113] 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), 2015.

[114] B. List, L.-W. Chen, K. Bali, and N. Thuerey. Differentiability in Unrolled Training of Neural Physics Simulators on Transient Dynamics, 2024. *arXiv:2402.12971.*

[115] R. Liu and J. Zou. The Effects of Memory Replay in Reinforcement Learning. In *56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 478–485, 2018.

[116] P. R. Lorenzo, J. Nalepa, M. Kawulok, L. S. Ramos, and J. R. Pastor. Particle Swarm Optimization for Hyper-Parameter Selection in Deep Neural Networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 481–488, 2017.

[117] D. J. Lucia, P. S. Beran, and W. A. Silva. Reduced-Order Modeling: New Approaches for Computational Physics. *Progress in Aerospace Sciences*, 40(1):51–117, 2004.

[118] F. Manrique de Lara and E. Ferrer. Accelerating High Order Discontinuous Galerkin Solvers using Neural Networks: 3D Compressible Navier-Stokes Equations. *Journal of Computational Physics*, 489, 2023.

[119] W. S. McCulloch and W. Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[120] N. McGreivy and A. Hakim. Weak Baselines and Reporting Biases Lead to Overoptimism in Machine Learning for Fluid-Related Partial Differential Equations. *Nature Machine Intelligence*, 6:1256–1269, 2024.

[121] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao. Large Language Models: A Survey, 2024. *arXiv:2402.06196.*

[122] M. Morimoto, K. Fukami, K. Zhang, A. G. Nair, and K. Fukagata. Convolutional Neural Networks for Fluid Flow Analysis: Toward Effective Metamodeling and Low Dimensionalization. *Theoretical and Computational Fluid Dynamics*, 35:633–658, 2021.

[123] F. Moukalled, L. Mangani, and M. Darwish. *The Finite Volume Method*. Springer International Publishing, first edition, 2016.

[124] K. P. Murphy. *Probabilistic Machine Learning: An Introduction*. MIT Press, 2022.

[125] G. Novati, H. Laroussilhe, and P. Koumoutsakos. Automating Turbulence Modelling by Multi-Agent Reinforcement Learning. *Nature Machine Intelligence*, 3:87–96, 2021.

[126] F. Ogoke, K. Meidani, A. Hashemi, and A. Barati Farimani. Graph Convolutional Networks Applied to Unstructured Flow Field Data. *Machine Learning: Science and Technology*, 2(4):045020, 2021.

[127] K. O'Shea and R. Nash. An Introduction to Convolutional Neural Networks, 2015. *arXiv:1511.08458*.

[128] D. W. Otter, J. R. Medina, and J. K. Kalita. A Survey of the Usages of Deep Learning for Natural Language Processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(2):604–624, 2021.

[129] M. Pak and S. Kim. A Review of Deep Learning in Image Recognition. In *2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)*, pages 1–3, 2017.

[130] D. Panchigar, K. Kar, S. Shukla, R. Mathew, U. Chadha, and S. K. Selvaraj. Machine Learning-Based CFD Simulations: A Review, Models, Open Threats, and Future Tactics. *Neural Computing and Applications*, 34:21677–21700, 2022.

[131] E. J. 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.

[132] S. C. Park, M. K. Park, and M. G. Kang. Super-Resolution Image Reconstruction: A Technical Overview. *IEEE Signal Processing Magazine*, 20:21–36, 2003.

[133] A. Parmar, R. Katariya, and V. Patel. A Review on Random Forest: An Ensemble Classifier. In *International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018*, pages 758–763, Cham, 2019. Springer International Publishing.

[134] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019.

[135] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[136] C. Pelletier, G. I. Webb, and F. Petitjean. Temporal Convolutional Neural Network for the Classification of Satellite Image Time Series. *Remote Sensing*, 11(5), 2019.

[137] P.-O. Persson and J. Peraire. *Sub-Cell Shock Capturing for Discontinuous Galerkin Methods*. 44th AIAA Aerospace Sciences Meeting and Exhibit, 2006. AIAA 2006-112.

[138] F. Piscaglia and F. Ghioldi. GPU Acceleration of CFD Simulations in OpenFOAM. *Aerospace*, 10(9), 2023.

[139] M. Plappert. keras-rl. `https://github.com/keras-rl/keras-rl`, 2016. *GitHub repository.*

[140] A. Probst, T. Knopp, C. Grabe, and J. Jägersküpper. HPC Requirements of High-Fidelity Flow Simulations for Aerodynamic Applications. In *Euro-Par 2019: Parallel Processing Workshops*, volume 11997, pages 375–387, Cham, 2020. Springer International Publishing.

[141] J. Rabault, F. Ren, W. Zhang, H. Tang, and H. Xu. Deep Reinforcement Learning in Fluid Mechanics: A Promising Method for both Active Flow Control and Shape Optimization. *Journal of Hydrodynamics*, 32(2):234–246, 2020.

[142] J. Romano and O. Baysal. *Convolutional-Neural-Network-Based Auto-Encoder for Synthetic Upscaling of Computational Fluid Dynamics Simulations*. AIAA SCITECH 2022 Forum, 2022. AIAA 2022-0186.

[143] S. Ruder. An Overview of Gradient Descent Optimization Algorithms, 2017. *arXiv:1609.04747.*

[144] C. Sabater, P. Stürmer, and P. Bekemeyer. Fast Predictions of Aircraft Aerodynamics Using Deep-Learning Techniques. *AIAA Journal*, 60(9):5249–5261, 2022.

[145] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. W. Battaglia. Learning to Simulate Complex Physics with Graph Networks. In *Proceedings of the 37th International Conference on Machine Learning*. JMLR.org, 2020.

[146] R. D. Sanhueza, S. H. Smit, J. W. Peeters, and R. Pecnik. Machine Learning for RANS Turbulence Nodeling of Variable Property Flows. *Computers & Fluids*, 255, 2023.

[147] 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, 2009.

[148] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation, 2018. *arXiv:1506.02438.*

[149] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms, 2017. *arXiv:1707.06347.*

[150] A. Schwarz, J. Keim, S. Chiocchetti, and A. Beck. A Reinforcement Learning Based Slope Limiter for Second-Order Finite Volume Schemes. *PAMM*, 23(1), 2023.

[151] M. Shin, M. Seo, K. Lee, et al. Super-resolution techniques for biomedical applications and challenges. *Biomedical Engineering Letters*, 14:465–496, 2024.

[152] J. Singh. An Improved Navier-Stokes Flow Computation of AGARD Case-10 Flow over RAE2822 Airfoil using Baldwin-Lomax Model. *Acta Mechanica*, 151:255–263, 2001.

[153] J. P. Slotnick, A. Khodadoust, J. J. Alonso, D. L. Darmofal, W. D. Gropp, E. A. Lurie, and D. J. Mavriplis. CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences. Technical report, NASA, 2014.

[154] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

[155] P. Spalart and S. Allmaras. *A One-Equation Turbulence Model for Aerodynamic Flows.* 30th Aerospace Sciences Meeting and Exhibit, 1992. AIAA 1992-439.

[156] W. Sutherland. LII. The Viscosity of Gases and Molecular Force. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, pages 507–531, 1893.

[157] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* A Bradford Book, Cambridge, MA, USA, 2018.

[158] J. P. Thomas, E. H. Dowell, and K. C. Hall. Three-Dimensional Transonic Aeroelasticity Using Proper Orthogonal Decomposition-Based Reduced-Order Models. *Journal of Aircraft*, 40(3):544–551, 2003.

[159] M. Tiberga, A. Hennink, J. L. Kloosterman, and D. Lathouwers. A High-Order Discontinuous Galerkin Solver for the Incompressible RANS Equations Coupled to the k-ϵ Turbulence Model. *Computers & Fluids*, 212, 2020.

[160] U. Trottenberg, C. Oosterlee, A. Schuller, and A. Brandt. *Multigrid.* Elsevier Science, 2001.

[161] U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid.* Academic Press, first edition, 2001.

[162] K. Um, R. Brand, Y. R. Fei, P. Holl, and N. Thuerey. Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers. In *Advances in Neural Information Processing Systems.* Curran Associates, Inc., 2020.

[163] R. Vinuesa and S. L. Brunton. Enhancing Computational Fluid Dynamics with Machine Learning. *Nature Computational Science*, 2022.

[164] 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, 2021.

[165] P. S. Volpiani, J.-B. Chapelier, A. Schwöppe, J. Jägersküpper, and S. Champagneux. Aircraft Simulations Using the New CFD Software from ONERA, DLR, and Airbus. *Journal of Aircraft*, 61(3):857–869, 2024.

[166] J.-X. Wang, J.-L. Wu, and H. Xiao. Physics-Informed Machine Learning Approach for Reconstructing Reynolds Stress Modeling Discrepancies based on DNS Data. *Phys. Rev. Fluids*, 2, 2017.

[167] L. Wang, Y. Fournier, J. F. Wald, and Y. Mesri. A Graph Neural Network-Based Framework to Identify Flow Phenomena on Unstructured Meshes. *Physics of Fluids*, 35(7):075149, 2023.

[168] Z. Wang. A Perspective on High-Order Methods in Computational Fluid Dynamics. *Science China Physics, Mechanics & Astronomy*, 59, 2015.

[169] J. A. White, H. Nishikawa, and R. A. Baurle. *Weighted Least-squares Cell-Average Gradient Construction Methods For The VULCAN-CFD Second-Order Accurate Unstructured Grid Cell-Centered Finite-Volume Solver.* AIAA Scitech 2019 Forum, 2019. AIAA 2019-0127.

[170] D. Wilcox. *Turbulence Modeling for CFD.* D C W Industries, third edition, 2006.

[171] R. J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8:229–256, 1992.

[172] F. D. Witherden and A. Jameson. *Future Directions in Computational Fluid Dynamics.* 23rd AIAA Computational Fluid Dynamics Conference, 2017. 2017-3791.

[173] X. Wu and P. Moin. Direct Numerical Simulation of Turbulence in a Nominally Zero-Pressure-Gradient Flat-Plate Boundary Layer. *Journal of Fluid Mechanics*, 630:5–41, 2009.

[174] G. G. Yan Wenhui, Yan Wei. Simulation of Transonic Viscous Flows Around RAE2822 Airfoil in GAO-YONG Compressible Turbulence Model. *Chinese Journal of Computational Physics*, 25(6):694–700, 2008.

[175] L. Yang and A. Shami. On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice. *Neurocomputing*, 415:295–316, 2020.

[176] M. Yang and Z. Xiao. Improving the k–ω–γ–Ar Transition Model by the Field Inversion and Machine Learning Framework. *Physics of Fluids*, 32(6), 2020.

[177] I. Yavneh. Why Multigrid Methods Are So Efficient. *Computing in Science & Engineering*, 8:12 – 22, 2006.

[178] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. WU. The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2022.

[179] X. Zhang, X. Guo, Y. Weng, X. Zhang, Y. Lu, and Z. Zhao. Hybrid MPI and CUDA Paralleled Finite Volume Unstructured CFD Simulations on a Multi-GPU System. *Future Generation Computer Systems*, 139:1–16, 2023.

[180] X.-L. Zhang, H. Xiao, X. Luo, and G. He. Ensemble Kalman Method for Learning Turbulence Models from Indirect Observation Data. *Journal of Fluid Mechanics*, 949, 2022.

[181] X.-L. Zhang, H. Xiao, X. Luo, and G. He. Combining Direct and Indirect Sparse Data for Learning Generalizable Turbulence Models. *Journal of Computational Physics*, 489, 2023.

[182] Z.-H. Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC, 1st edition, 2012.

[183] I. R. J. Zwaan. LANN Wing Pitching Oscillations. *Compendium of Unsteady Aerodynamic Measurements, AGARD-R-702 Addendum No. 1*, 1982.

[184] X. Álvarez Farré, A. Gorobets, and F. X. Trias. A Hierarchical Parallel Implementation for Heterogeneous Computing. Application to Algebra-Based CFD Simulations on Hybrid Supercomputers. *Computers & Fluids*, 214, 2021.